

USER MANUAL

Turbo PMAC Ladder

Programming Development Environment

3A0-LADDER-xUxx

January 29, 2003



Copyright Information

© 2003 Delta Tau Data Systems, Inc. All rights reserved.

This document is furnished for the customers of Delta Tau Data Systems, Inc. Other uses are unauthorized without written permission of Delta Tau Data Systems, Inc. Information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support

Phone: (818) 717-5656

Fax: (818) 998-7807

Email: support@deltatau.com

Website: <http://www.deltatau.com>

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

Table of Contents

INTRODUCTION	1
STYLES AND SYMBOLS	1
REQUIREMENTS	1
<i>Hardware</i>	<i>1</i>
<i>Software</i>	<i>2</i>
INSTALLATION	2
STARTING TURBO PMAC LADDER	3
QUICK GETTING STARTED	4
CREATING THE MINIMUM PROJECT.....	5
PROJECT BROWSER	5
CREATE PROJECT	6
CREATING A PROGRAM IN YOUR PROJECT	7
USING THE CONTROL X EDITOR	10
<i>Variable Editor</i>	<i>10</i>
<i>Instruction Editor.....</i>	<i>11</i>
COMPILE / BUILD PROJECT	15
GO ONLINE TO RUN & TEST PROGRAM IN THE PC.....	15
STARTING AND STOPPING THE PROGRAM	16
WATCHING VARIABLES & SETTING VARIABLES.....	17
ONLINE CONTROLX EDITOR AND POWER/DATA FLOW	17
GO ONLINE TO RUN & TEST PROGRAM IN TURBO PMAC.....	17
<i>Editing a Resource.....</i>	<i>17</i>
<i>Go Online to PMAC.....</i>	<i>18</i>
FUNDAMENTALS OF LADDER, FUNCTION BLOCK AND SFC.....	20
LADDER DIAGRAM (LD)	20
<i>Programming in Ladder Diagram.....</i>	<i>20</i>
FUNCTION BLOCK DIAGRAM (FBD)	21
<i>Use of Function Blocks.....</i>	<i>21</i>
SEQUENTIAL FUNCTION CHART (SFC).....	22
<i>Elements of the Sequential Function Chart</i>	<i>22</i>
THE PROJECT BROWSER.....	24
START OF THE PROJECT BROWSER	25
MANAGING PROJECTS	25
<i>Create a new project.....</i>	<i>25</i>
<i>Open an existing project.....</i>	<i>26</i>
<i>Copy a project.....</i>	<i>27</i>
<i>Create a backup copy</i>	<i>27</i>
<i>Restore a project.....</i>	<i>28</i>
<i>Deleting projects.....</i>	<i>29</i>
<i>Rename project</i>	<i>30</i>
<i>Add Files.....</i>	<i>31</i>
THE PROJECT TREE	31
<i>Basics of the Project Tree.....</i>	<i>31</i>
<i>The Project Directory.....</i>	<i>32</i>
<i>Libraries</i>	<i>33</i>
WORK WITH MODULES AND TYPE DECLARATION FILES.....	34
<i>Creating Modules</i>	<i>34</i>
<i>Editing Modules.....</i>	<i>35</i>
<i>Copy.....</i>	<i>35</i>
<i>Move</i>	<i>36</i>

<i>Global Resource Variables</i>	36
<i>Type definitions</i>	38
RESOURCES	39
<i>Creation of a Resource</i>	40
<i>Edit a Resource</i>	40
<i>Adding a Task</i>	41
<i>Adding [direct] Global Variables</i>	42
<i>Adding Type Definitions</i>	43
<i>Generate Executable Code</i>	43
TEST AND COMMISSIONING	45
<i>Going Online</i>	45
<i>Starting and Stopping the Program</i>	45
<i>Watching Variables</i>	45
<i>Set Variables</i>	46
<i>The Online-Editor</i>	46
<i>Hardware Info</i>	48
<i>Resource Info</i>	48
CONFIGURATION OF THE PROJECT MANAGER	49
<i>Settings</i>	49
<i>Print Setup</i>	50
CONTROL X EDITOR	51
VARIABLE EDITOR	51
OVERVIEW OF THE DECLARATION TYPE	52
INSTRUCTION EDITOR	53
<i>Instructions</i>	54
ADDING/EDITING LADDER PROGRAM	55
<i>Inserting New Network</i>	55
<i>Inserting a Logical AND</i>	55
<i>Inserting a Logical OR</i>	55
<i>Inserting a Parallel Coil or JMP</i>	55
<i>Inserting a Function Block</i>	55
<i>Inserting a Function</i>	56
<i>Inserting a Variable</i>	56
<i>Saving and checking the Syntax of Your Program</i>	56
ADDING/EDITING SFC PROGRAM	57
<i>Steps and Initial Steps</i>	57
<i>Transitions</i>	57
<i>Jumps</i>	58
<i>Editing of the (Step/Transition) Program Blocks</i>	58
<i>The Structure of an IL-Line</i>	60
<i>Syntax control</i>	61
PMAC LADDER – EXAMPLE	63
OPEN PROJECT	63
COMPILE / BUILD PROJECT	63
GO ONLINE	63
STARTING AND STOPPING THE PROGRAM	64
WATCHING VARIABLES	64
WATCHING POWER FLOW	65
CROSSREFERENCE	65
EXAMPLE OUTPUT	66
SFC PROGRAM - EXAMPLE	67
SET THE ACTIVE RESOURCE	67
COMPILE / BUILD PROJECT	67

Go ONLINE	67
STARTING AND STOPPING THE PROGRAM	68
WATCHING VARIABLES	69
WATCHING POWER FLOW	69
PMAC LADDER REFERENCE SECTION	70
PMAC LADDER LD FUNCTIONS	70
STANDARD ARITHMETIC FUNCTIONS	71
ABS_LD24	71
ADD_LD24	72
SUB_LD24	72
MUL_LD24	72
DIV_LD24	73
MOD_LD24	73
NEG_LD24	74
ABS_LD48	74
ADD_LD48	74
SUB_LD48	74
MUL_LD48	75
DIV_LD48	75
MOD_LD48	75
NEG_LD48	76
STANDARD LOGICAL FUNCTIONS	77
AND_LD24	77
OR_LD24	77
EOR_LD24	77
NOT_LD24	78
ROL_LD24	78
ROR_LD24	78
SHL_LD24	78
SHR_LD24	79
AND_LD48	79
OR_LD48	79
EOR_LD48	80
COMPLEX ARITHMETIC FUNCTIONS	80
SIN_LD48	80
COS_LD48	81
TAN_LD48	81
ASIN_LD48	81
ACOS_LD48	82
ATAN_LD48	82
ATAN2_LD48	82
SQRT_LD48	82
EXP_LD48	83
LN_LD48	83
LOG_LD48	83
TRUNC_LD48	84
LIMIT FUNCTIONS	84
MAX_LD24	84
MIN_LD24	85
MAX_LD48	85
MIN_LD48	85
ASSIGNMENT FUNCTIONS	86
MOVE_LD24	86
MOVE_LD48	86
ARITHMETIC COMPARISON FUNCTIONS	86

<i>EQ_ILD24</i>	86
<i>GE_ILD24</i>	87
<i>GT_ILD24</i>	87
<i>LE_ILD24</i>	87
<i>LT_ILD24</i>	88
<i>NE_ILD24</i>	88
<i>EQ_ILD48</i>	88
<i>GE_ILD48</i>	88
<i>GT_ILD48</i>	89
<i>LE_ILD48</i>	89
<i>LT_ILD48</i>	89
<i>NE_ILD48</i>	90
DATA TYPE CONVERSION FUNCTIONS	90
<i>LD24_TO_LD48</i>	90
<i>LD48_TO_LD24</i>	90
<i>LD24_TO_INT</i>	91
<i>INT_TO_LD24</i>	91
<i>GREY4_TO_LD24</i>	91
GET TIME FUNCTIONS.....	92
<i>GetTime_LD48</i>	92
PMAC LADDER LD & SFC FUNCTION BLOCKS	92
SPECIAL PMAC FUNCTION BLOCKS	92
<i>PMAC_CMD_STR</i>	92
<i>PMAC_CMDL_STR</i>	93
<i>PMAC_DISP_VAR</i>	94
<i>PMAC_DISP_STR</i>	94
<i>PMAC_LOCK</i>	94
<i>PMAC_UNLOCK</i>	95
<i>PMAC_SET_PHASE</i>	95
<i>PMAC_CYCLE_TIME</i>	95
IEC LADDER & SFC FUNCTION BLOCKS	96
STANDARD IEC FUNCTION BLOCKS	96
<i>F_TRIG</i>	96
<i>R_TRIG</i>	96
<i>RS</i>	96
<i>SR</i>	97
<i>TOF</i>	97
<i>TON</i>	98
<i>TP</i>	98
<i>CTD</i>	99
<i>CTU</i>	99
<i>CTUD</i>	100
PMAC LADDER SFC INSTRUCTION LIST (IL) FUNCTIONS	101
STANDARD ARITHMETIC FUNCTIONS	102
<i>ABS_IL24 : INT24</i>	102
<i>ADD_IL24 : INT24</i>	103
<i>SUB_IL24 : INT24</i>	103
<i>MUL_IL24 : INT24</i>	103
<i>DIV_IL24 : INT24</i>	103
<i>MOD_IL24 : INT24</i>	103
<i>NEG_IL24 : INT24</i>	104
<i>ABS_IL48 : REAL48</i>	104
<i>ADD_IL48 : REAL48</i>	104
<i>SUB_IL48 : REAL48</i>	104
<i>MUL_IL48 : REAL48</i>	104

<i>DIV_IL48 : REAL48</i>	105
<i>MOD_IL48 : REAL48</i>	105
<i>NEG_IL48 : REAL48</i>	105
STANDARD LOGICAL FUNCTIONS	105
<i>AND_IL24 : INT24</i>	105
<i>OR_IL24 : INT24</i>	105
<i>EOR_IL24 : INT24</i>	106
<i>NOT_IL24 : INT24</i>	106
<i>ROL_IL24 : INT24</i>	106
<i>ROR_IL24 : INT24</i>	106
<i>SHL_IL24 : INT24</i>	106
<i>SHR_IL24 : INT24</i>	107
<i>AND_IL48 : REAL48</i>	107
<i>OR_IL48 : REAL48</i>	107
<i>EOR_IL48 : REAL48</i>	107
COMPLEX ARITHMETIC FUNCTIONS	108
<i>SIN_IL48 : REAL48</i>	108
<i>COS_IL48 : REAL48</i>	108
<i>TAN_IL48 : REAL48</i>	108
<i>ASIN_IL48 : REAL48</i>	108
<i>ACOS_IL48 : REAL48</i>	109
<i>ATAN_IL48 : REAL48</i>	109
<i>ATAN2_IL48 : REAL48</i>	109
<i>SQRT_IL48 : REAL48</i>	109
<i>EXP_IL48 : REAL48</i>	109
<i>LN_IL48 : REAL48</i>	109
<i>LOG_IL48 : REAL48</i>	110
<i>TRUNC_IL48 : REAL48</i>	110
LIMIT FUNCTIONS	110
<i>MAX_IL24 : INT24</i>	110
<i>MIN_IL24 : INT24</i>	110
<i>MAX_IL48 : REAL48</i>	111
<i>MIN_IL48 : REAL48</i>	111
ARITHMETIC COMPARISON FUNCTIONS	111
<i>EQ_ILD24 : BOOL</i>	111
<i>GE_ILD24 : BOOL</i>	111
<i>GT_ILD24 : BOOL</i>	112
<i>LE_ILD24 : BOOL</i>	112
<i>LT_ILD24 : BOOL</i>	112
<i>NE_ILD24 : BOOL</i>	113
<i>EQ_ILD48 : BOOL</i>	113
<i>GE_ILD48 : BOOL</i>	113
<i>GT_ILD48 : BOOL</i>	113
<i>LE_ILD48 : BOOL</i>	114
<i>LT_ILD48 : BOOL</i>	114
<i>NE_ILD48 : BOOL</i>	114
DATA TYPE CONVERSION FUNCTIONS	114
<i>LD24_TO_IL48 : REAL48</i>	114
<i>LD48_TO_IL24 : INT24</i>	115
<i>LD24_TO_INT : INT</i>	115
<i>INT_TO_LD24 : INT24</i>	115
<i>GREY4_TO_LD24 : INT24</i>	115
GET TIME FUNCTIONS	116
<i>GetTime_IL48 : REAL48</i>	116
PMAC LADDER INPUT & OUTPUT	117
<i>IEC-1131 Input - Output Qualifier</i>	117

PMAC Address Qualifiers.....117

I/O Process Image Data Type Qualifier119

Example PMAC Input & Output.....119

INTRODUCTION

PMAC Ladder is a programming development environment using the IEC-61131 standard American Ladder Logic and Sequential Function Chart languages. The reduced price version, the Monitor, allows only viewing and setting program variables, and downloading of IEC programs.

Features of the program...

- IEC-61131 languages Ladder Logic with Function Blocks and Sequential Function Charts (SFC).
- Standard IEC data types (INT, TIME, BOOL, DWORD) and Counter - Timer Function Blocks.
- All PMAC PLC commands are available in the IEC environment. These include the string commands used for motor jogging and homing (“&lbr #lj+ hm”, etc.).
- New PMAC data types INT24 (integer 24 bit) and REAL48 (PMAC float) for maximum computational power.
- Input & Output support for all PMAC hardware (I, M, P, Q variables), MACRO and direct access to thumbwheel , X, Y, and DPR memory.
- Testing environment that allows starting, stopping, downloading, setting and monitoring of variables in a watch window or logic power flow.
- Simulation mode for developing your program on the PC without PMAC hardware.
- Printing of Ladder and SFC programs with user defined forms that include bit maps.
- PMAC serial port dedicated for IEC communication.
- Fast PLC execution mode (Native code). Similar to PMAC’s PLCCs.
- Debug cross-reference for easy access to where variables are in a program.

The Free DEMO version will give you 30 days to evaluate PMAC Ladder in the simulation mode on a PC.

Styles and Symbols

The Application name OPEN PCS and TURBO PMAC Ladder are interchangeable and specify the same program. The notation Project-->New is used to denote the selection of item **New** in menu **Project**. The arrow --> is used to mark individual steps you should follow.

Requirements

Hardware

As a programming system, you will need an IBM compatible PC with

- At least an 80486 microprocessor (we recommend at least a Pentium 166) and a CD-ROM-drive
- A hard disk drive with at least 30 Mbytes of free space.
- At least 8 Mbytes of RAM (we recommend 32MB).
- VGA graphics card.
- Mouse
- Turbo PMAC
 - Auxiliary Serial Port (Option 9T)
 - Additional memory (Option 5x3)
 - Bootstrap version 1.08 or greater
 - Firmware version 1.938 or greater
 - PMAC Ladder firmware 3.4 or greater

Software

- Windows 98/2000 or Windows NT 4.0 operating system.
- Turbo PMAC Ladder program.

Installation

In the PMAC Ladder CD start the installation as you would most Windows programs by launching **SETUP.EXE**.

If you encounter the below installation windows, do the following

- “A previous version exists! Setup will over write this version. Do you want to continue?” Select **Yes**.
- “Do you want to make a backup of the OpenPC’s configuration directory?” Select **No**.
- Select **Ok** when you are asked to enter a **License** and don’t enter anything..
- “Do you want to copy user specific setting from ...?” Select **No**.
- If asked if “you have a disk with hardware drivers?” select **No**.
- “Are you sure you want to install PMAC hardware components ...?” Select **Yes**.

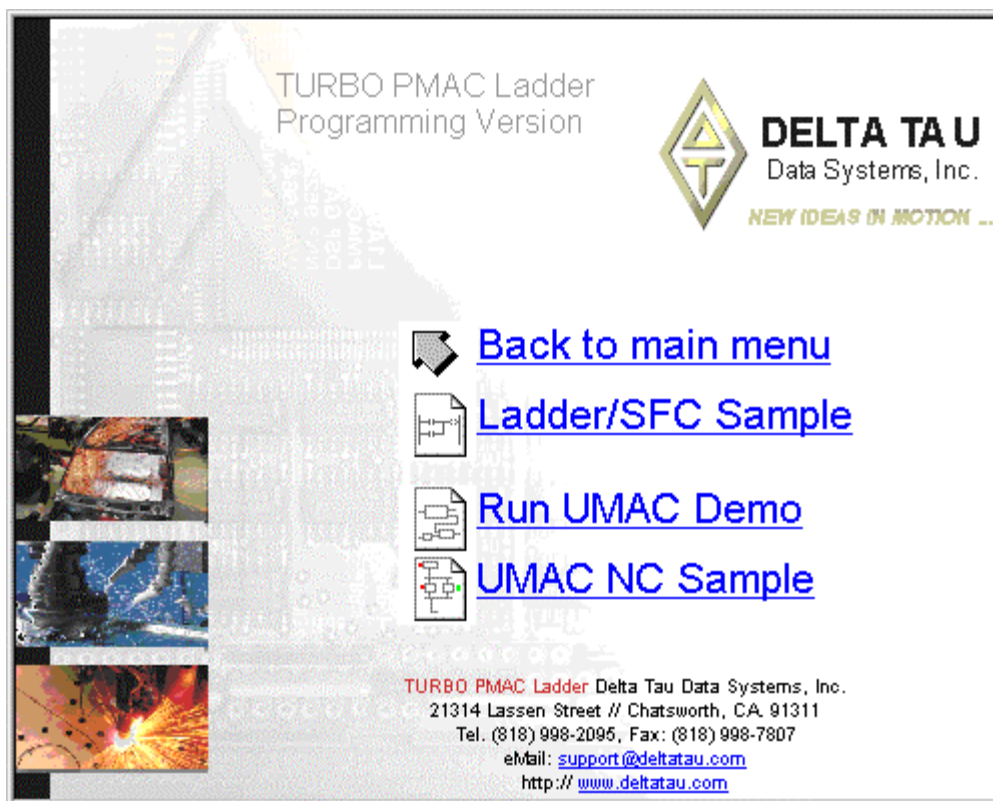
The successful installation can be verified by the successful program startup.

Starting Turbo PMAC Ladder

Select Open PCS icon from the desktop or Start Windows and choose “start → programs → *OpenPCS* → *OpenPCS*” in the start-menu. The following window will be opened. Choose one of the four selections. The sample projects include ones that run a four axes UMAC Demo box and a six-axes UMAC Demo box with an Advantage 810.



Selecting the ‘Sample Projects’ will bring up a selection of sample programs. After selecting any of these or any on the previous window, the Project Browser will be displayed.



Quick Getting Started

See **README.PDF** for getting started with the already created project **Ladder/SFC Sample**.

CREATING THE MINIMUM PROJECT

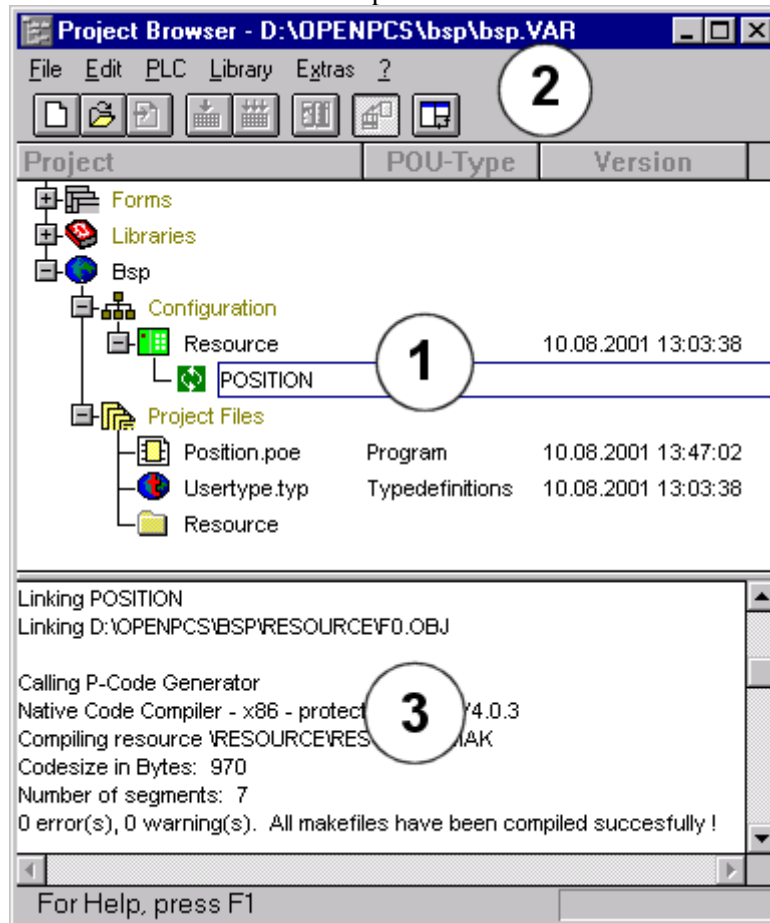
Project Browser

The tool **Project Browser** is used to manage projects and the files belonging to the project.

With the Project Browser you can structure your work into projects. A great advantage of OpenPCS is the reuse of blocks in other projects. Please note the following hint:

- Avoid creating several OpenPCS-projects for related work. One machine, a network of machines or even one plant might be one project.
- Use separate projects for unrelated work; this increases your overview and prevents you from mistakenly modifying wrong code.

The project management window consists of three parts:

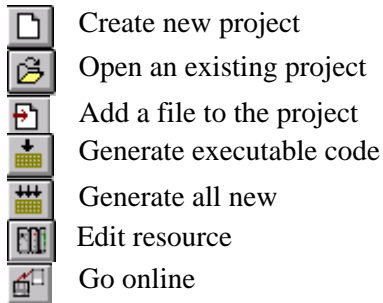


1. **The project tree.** Here you can see all information concerning the project in the graphical form of a software tree. The column headed POU-Type informs you about the used program-organization-unit (program, functionblock, etc.). The column Version shows the last saving of a pou-file.

The width of the columns can be arranged individually and will be saved at the closing the Project Browser.

2. **The menu panel and the toolbar.**

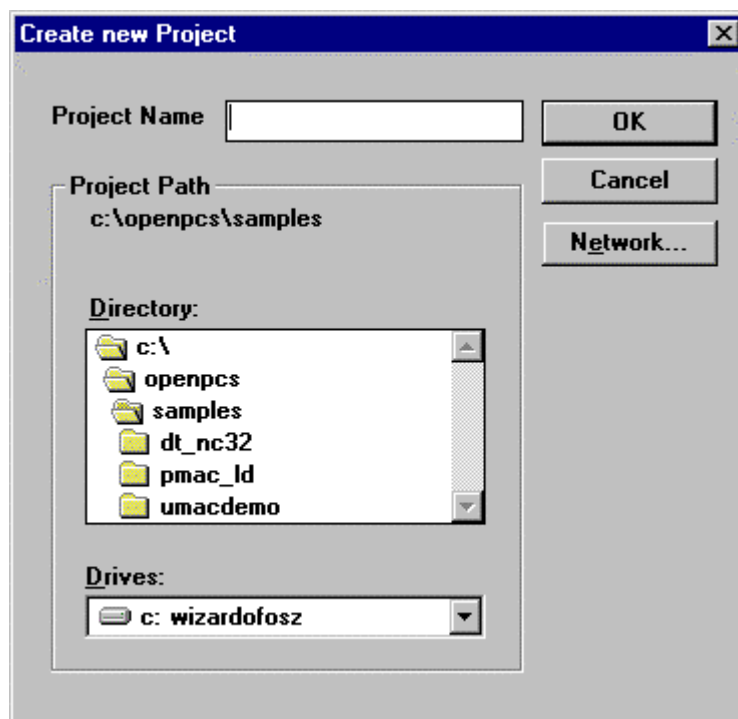
The toolbar:



3. **The output window.** Output of the compilation or the online operation are shown here. This manual explains the functionality of the project browser.

Create Project

Create a **NEW** project by selecting *File->Project → New*. A dialog box will be shown to prompt you for the name and directory path of this new project



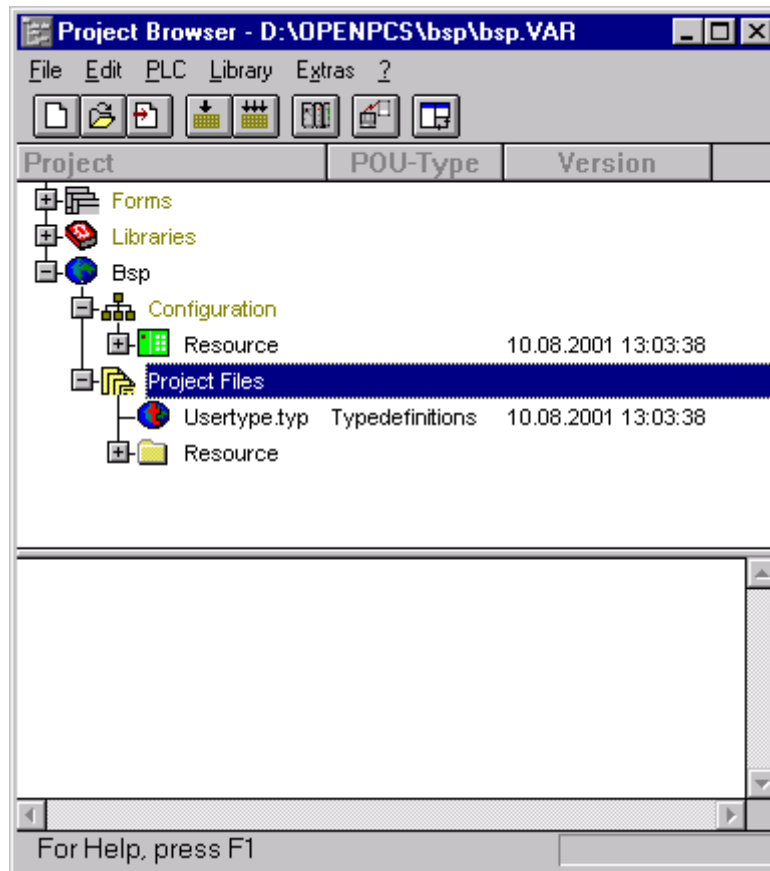
You may use a path on a non-local, networked drive, but you cannot use this to work with more than one copy of Open PCS at once with the same project!

The length of a project path is limited. Try to use as few characters for the directory name as possible.

You will see an error message if the name is more than eight characters long.

Select directory C:\OpenPCS and enter **bsp** as the project name:

- bsp
- Close the dialogbox with **OK**. The path **c:\OpenPCS\bsp** will be created and the minimum project structure will be shown:



With a new, empty project only the three default entries **Forms**, **Libraries** and the project (in this case **Bsp**) will exist.

The browser automatically installs the PC simulation resource under the branch **Configuration** so you can test the created POU's on the SmartSIM32.

In addition a file **Uertype.typ** exists under the branch **Project Files**.

During the course of this manual we will create more entries to be shown in this tree.

Creating a Program in your Project

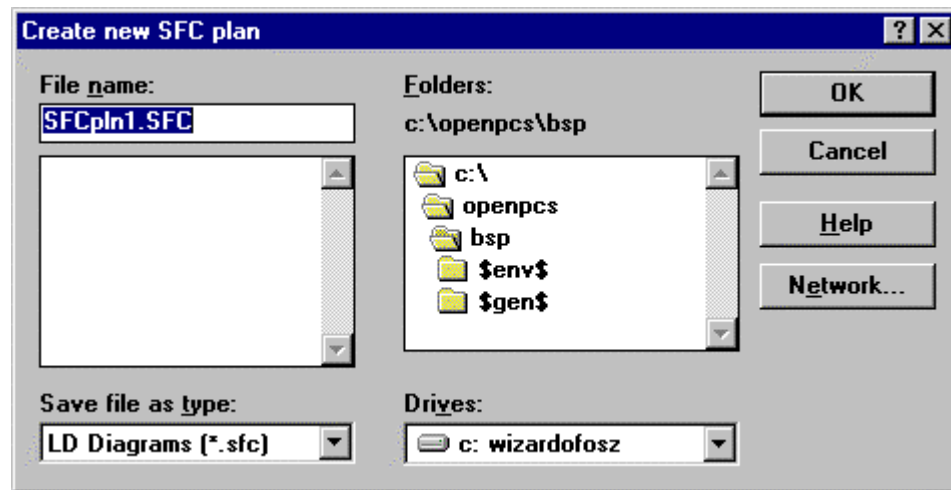
The programs that can be created for the existing project or for the new project are –

SFC Program (Sequential Function Chart), and LD Program (Ladder program)

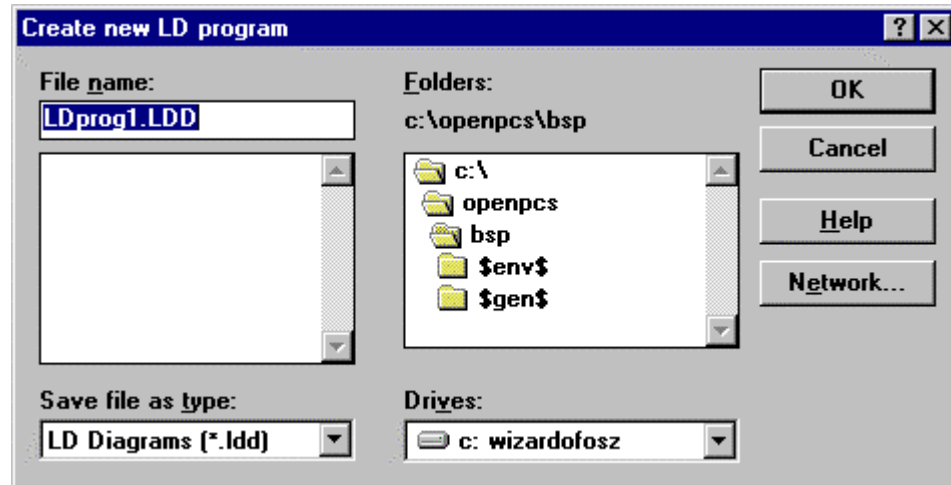
To create a new program,

- Select the action **File → New → Program**.
- Choose the type of the program that you want to create. The keywords are **SFC** or **LD Program**.
-

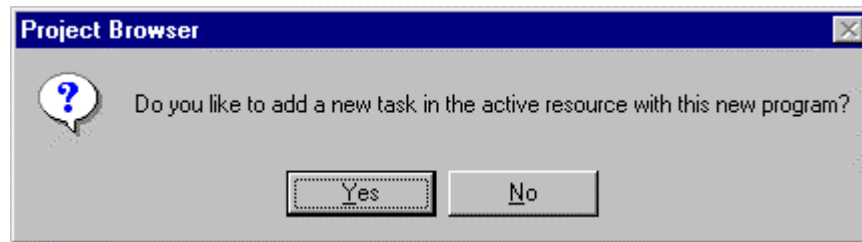
The following dialog box is opened if **SFC Program** is selected



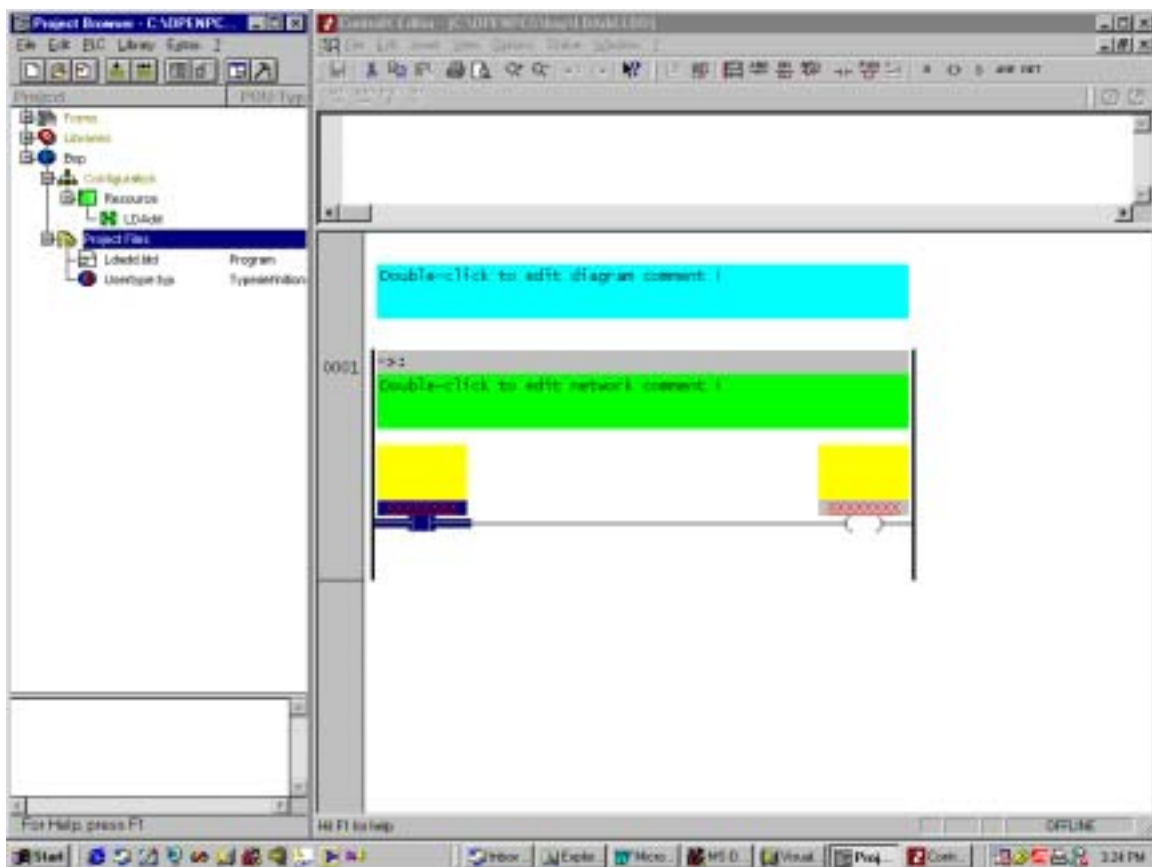
The following dialog box is opened if **LD Program** is selected



In File name change **LDprog1.LDD** to **LDAdd.LDD** and select the **OK** button to save under **c:\openpcs\bsp** project. You will see the following window. Select the **Yes** button.



The LDAdd will be added to the project tree under **Project Files** and **Resource** and the Controlx Editor for LDAdd.LDD will be opened.



Using the Control X Editor

The Control X Editor is used to write a Ladder program, Function Block or SFC program.

The Control X Edit Framework consists of two independent windows. The first window is divided into the **variable editor** (upper section) and the **instruction editor**. All the variables are to be declared in the Variable Editor. The instruction editor is used to write the program segments.

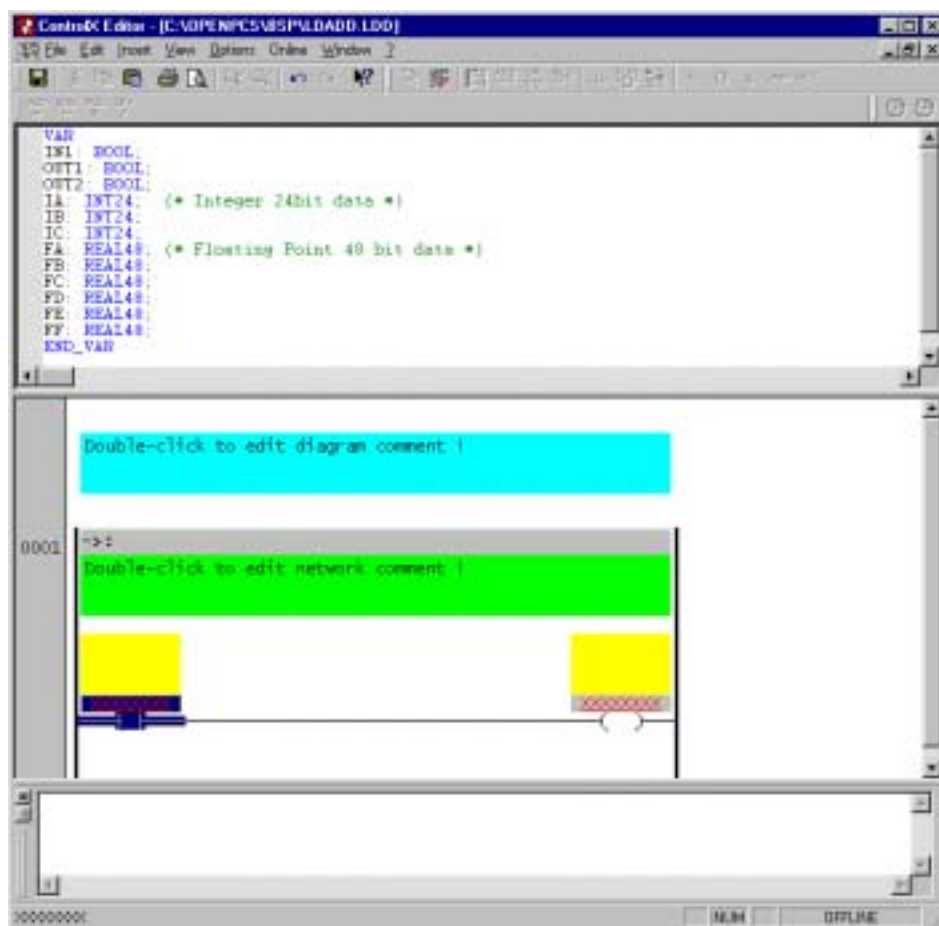
The **output window** is a part of the Control X editor framework. It displays the results of the syntax check or the errors from the compilation. The Syntax Check can be done by Alt+F10 key or by selecting **File → Check Syntax**.

Variable Editor

All the variables used by Ladder or Function Block or SFC program elements must be declared before compilation.

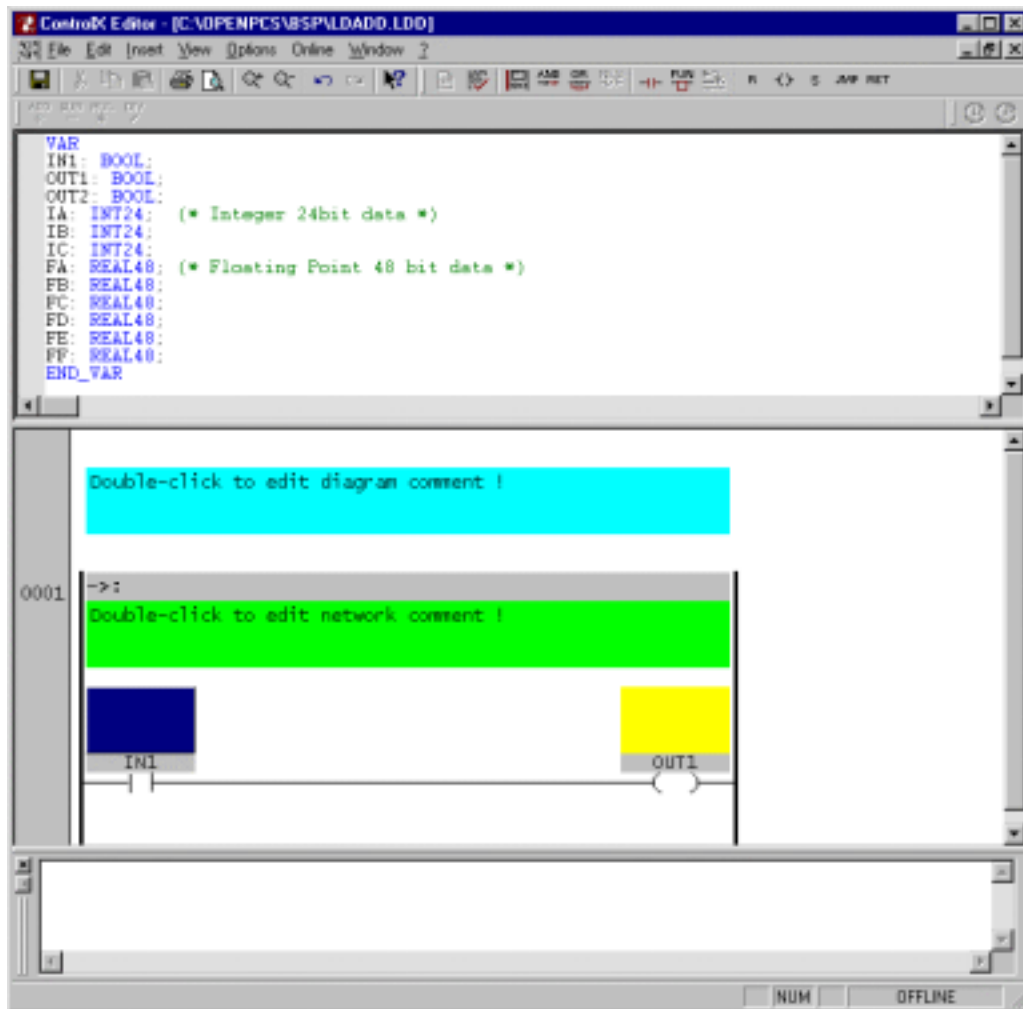
The variables of each type must be declared within a separate declaration block. It is advisable to separate the different components of a declaration line with tabs. Each declaration block is introduced and closed with a certain key word (shown in Blue), e.g. **VAR** and **END_VAR** for local variables. A comment begins with an opening bracket and an asterisk "(*" and ends with an asterisk and a closing bracket "*")". Comments are displayed in the colour green.

So let's do that now by entering the variables shown below:

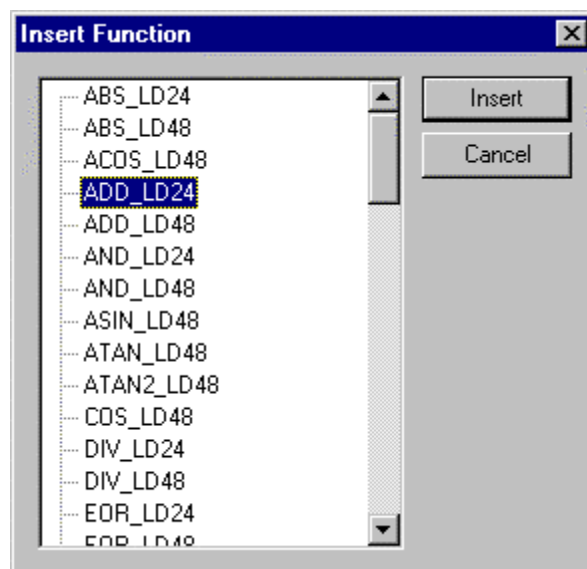


Instruction Editor

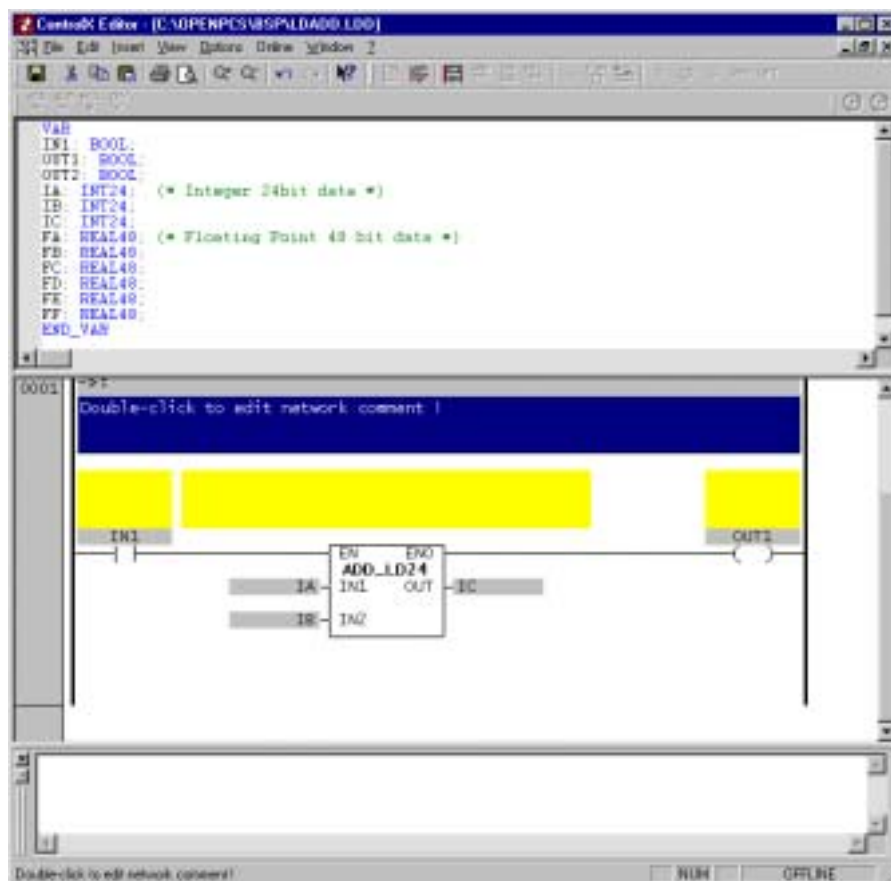
This is where the program is entered. Use the right mouse to select the contact and coil and then select **Insert Variable**. Then insert the **BOOL** variables IN1 and OUT1 as shown below:



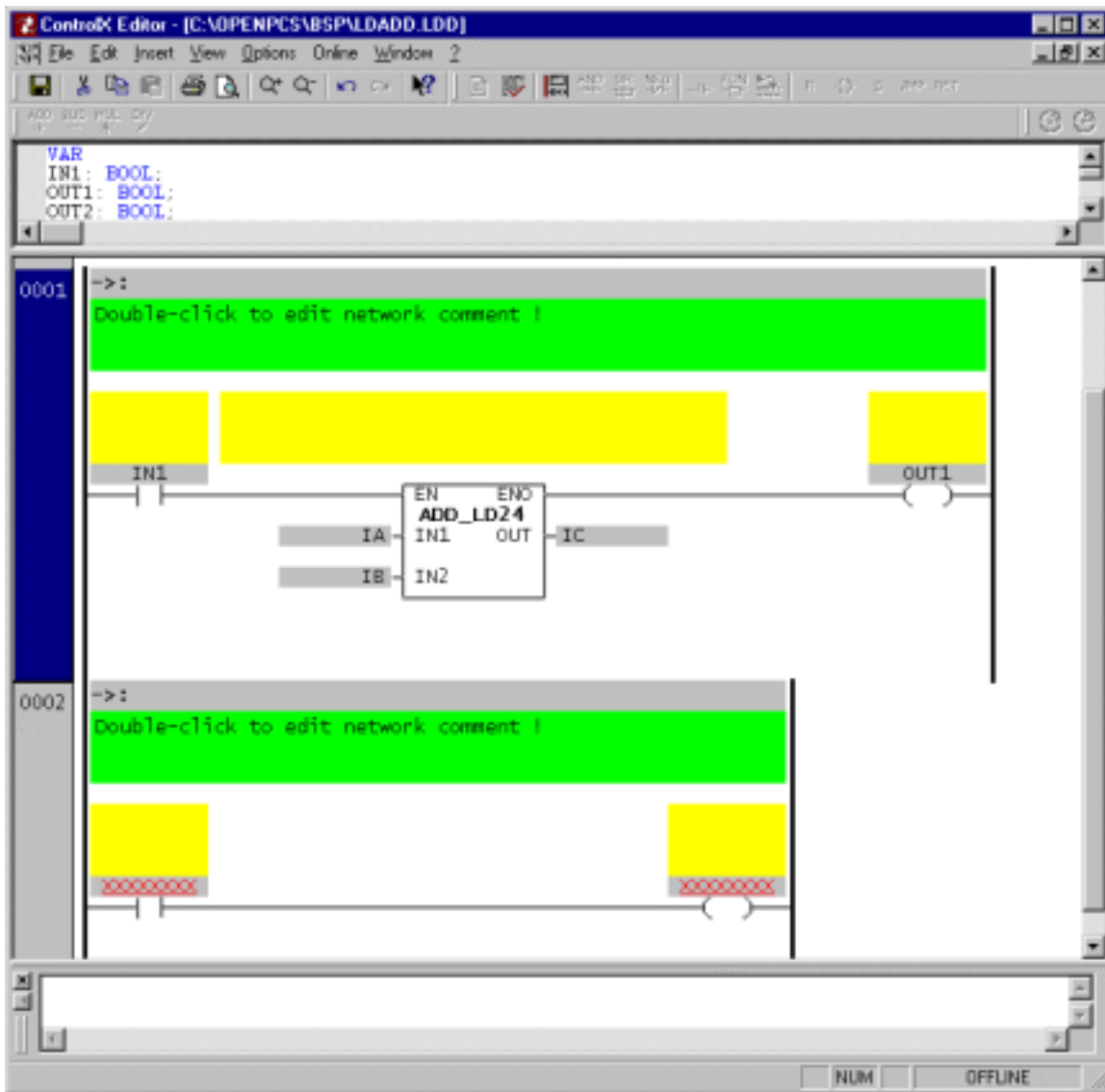
Next, right- mouse select the line between IN1 and OUT1, and select **Insert Function** and insert the **ADD_LD24** function as shown below:



Next select the **ADD_LD24** variables with a right mouse selecting the **Insert Variables IA, IB, IC** as shown below:



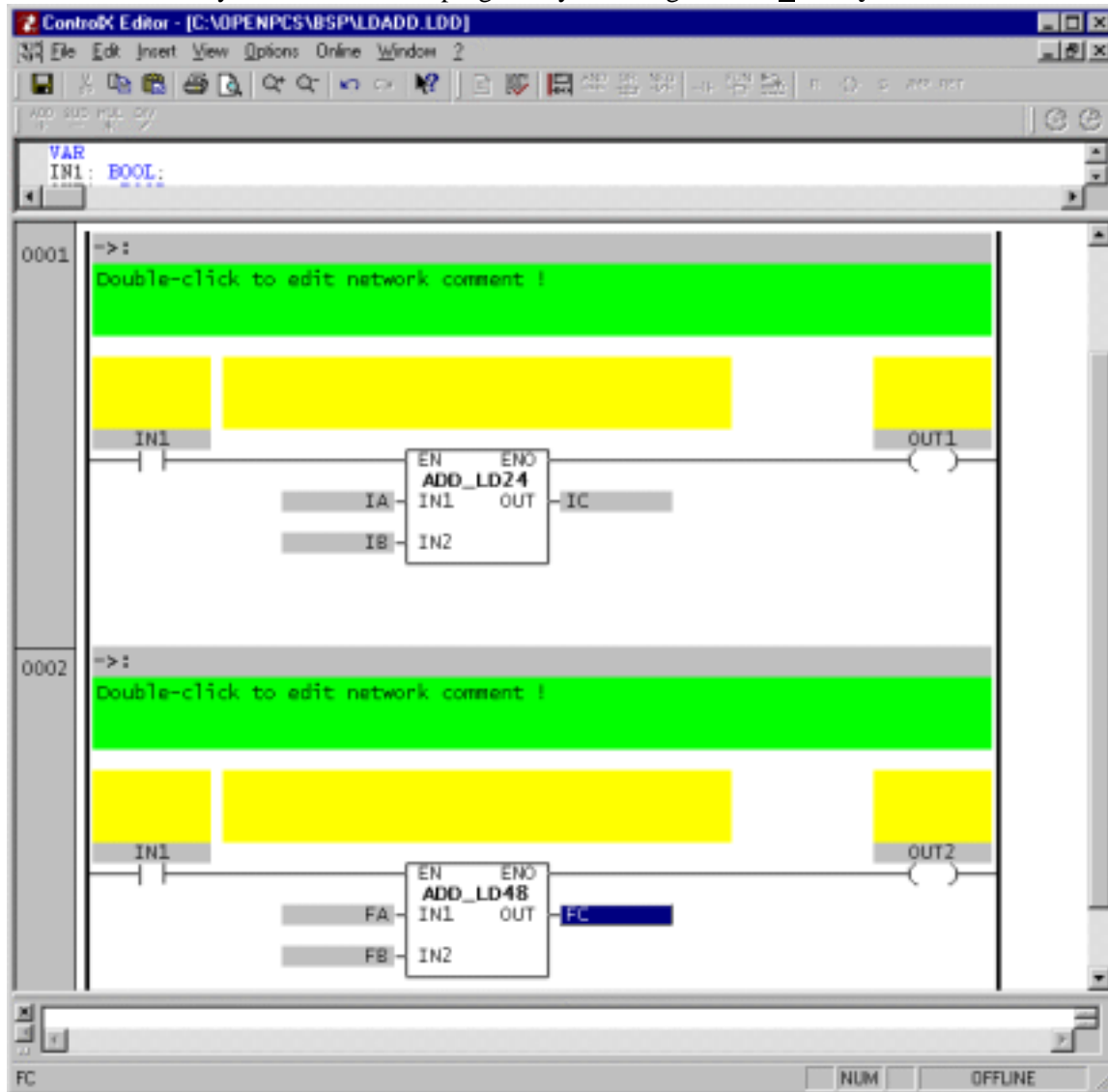
Now we want to make another **Network**, so right-mouse click on **0001** and **Insert Network**. You should have the results below:



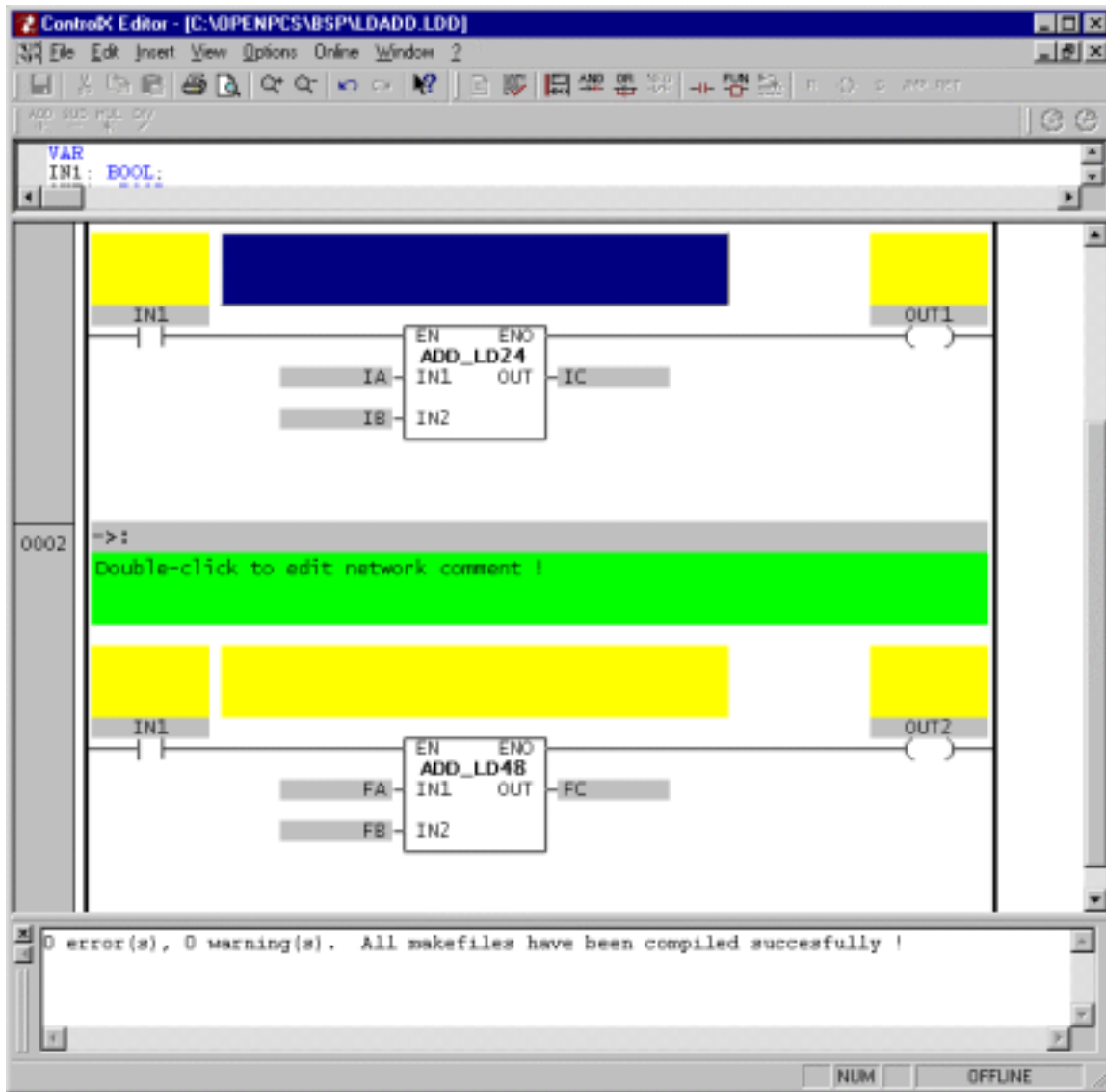
Now repeat with similar steps of **Insert Variables** and **Insert Function (ADD_LD48)** as shown below:

Output Window

Now we will do a syntax check on our program by selecting **File->Check Syntax**. You will



be asked "Do you want to save changes?" Select **Ok**. You should get the following results:



You have successfully created a ladder program. Now we will compile the project.

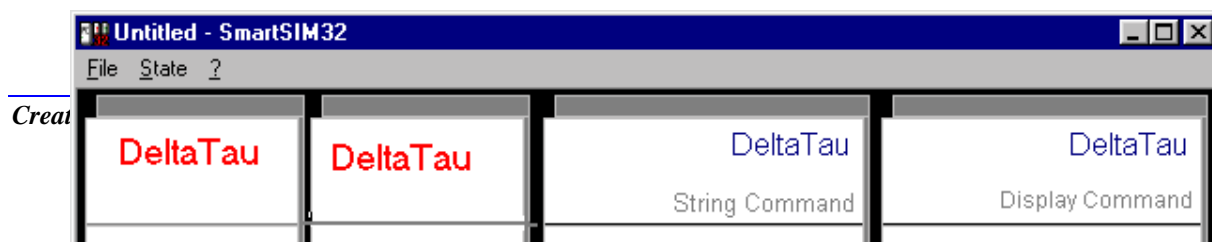
Compile / Build Project

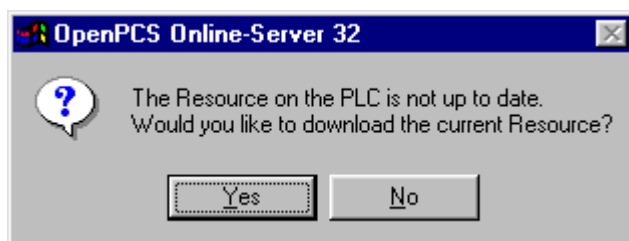
Under the **Project Browser**, highlight **Configuration\Resource** and in the menu select **PLC → Build All**. In the **Browser Output** window you should see the compilation of the program showing “0 error(s) and 0 warning(s).”

Go Online to Run & Test Program in the PC

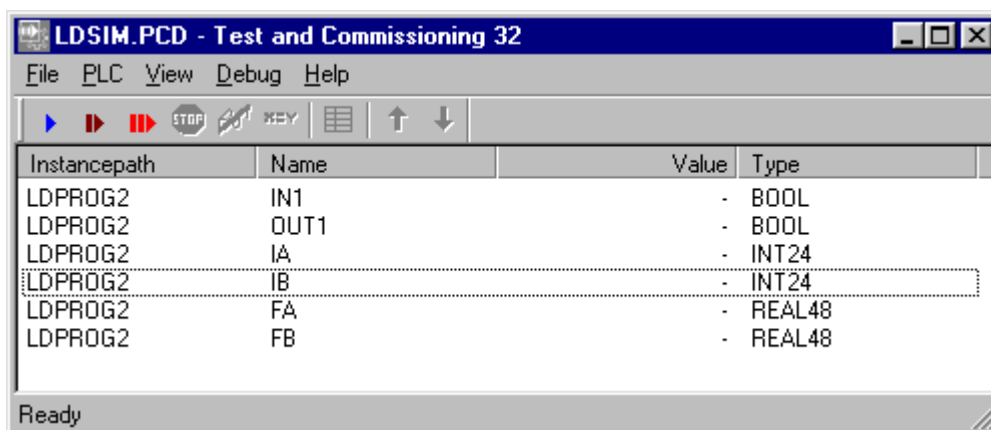
Under the **Project Browser**, highlight **Configuration\Resource** and in the menu select **PLC → Online**.

You should see the following two windows plus a **Test and Commissioning 32** window. Select **Yes** to download the program to run a simulation of the program in the PC.









In the **Project Browser Configuration\Resource\LDADD** tree double mouse click on IN1, OUT1, IA, IB, FA and FB. In the **Test and Commissioning Window** you should see something like the following:



Starting and Stopping the Program

Press the Coldstart **Blue Arrow** button to start the program from Test and Commission 32 window.

- | | | |
|---|------------------------|--|
|  | PLC → STOP | Select PLC → Stop to immediately stop the program |
|  | PLC → Coldstart | Select PLC → Coldstart to perform a cold start. All variables will be initialized to the initial value as programmed. |
|  | PLC → Warmstart | Select PLC → Warmstart to initialise all normal variables, but keep the values of all variables programmed as RETAIN. |
|  | PLC → Hotstart | Select PLC → Hotstart to continue execution where |

it stopped, not initializing any variable.

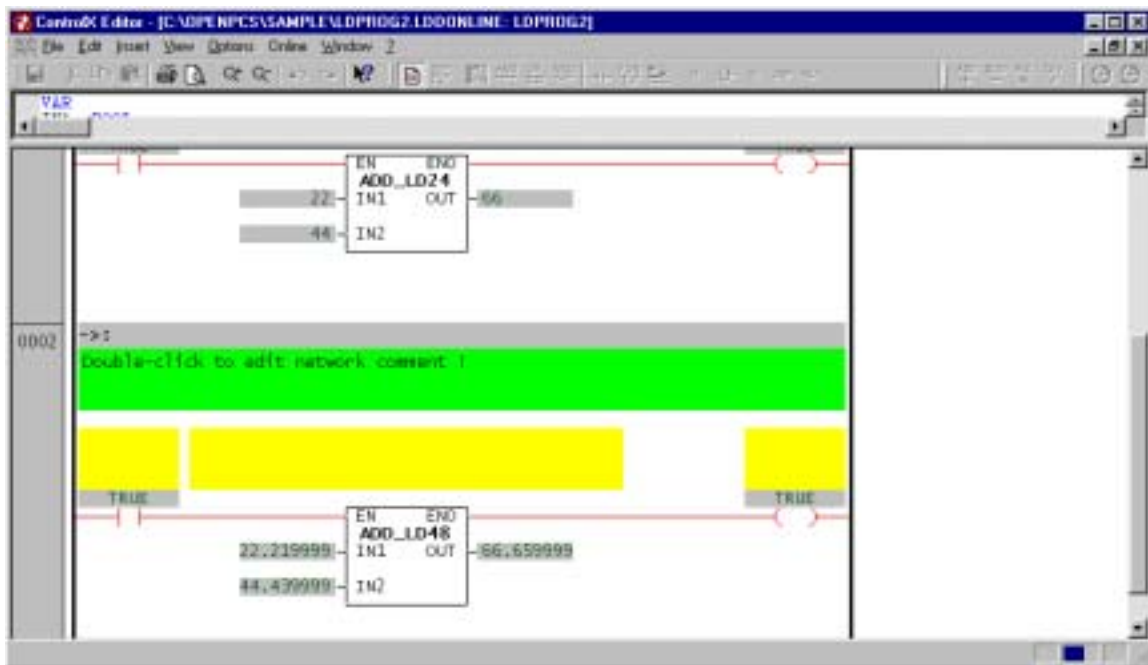
You should see IN1 and OUT1 go to FALSE and the other variables go to zero.

Watching Variables & Setting Variables

Go to the **Test and Commissioning Window** and double click **IN1** and set it to **1**. You should now see OUT1 go to TRUE. Do the same for **IA** and **IB** setting them to 22 and 44 and **FA** and **FB** to 22.22 and 44.44.

Online ControlX Editor and Power/Data Flow

Highlight **Configuration\Resource\LDADD**, then right-mouse click to **Open** the **LDADD** program for power/data observation. You should see **IA=22, IB=44, IC = 66** and **FA=22.22, FB=44.44** and **FC = 66.66** in the **ControlX Editor** window like that shown below.

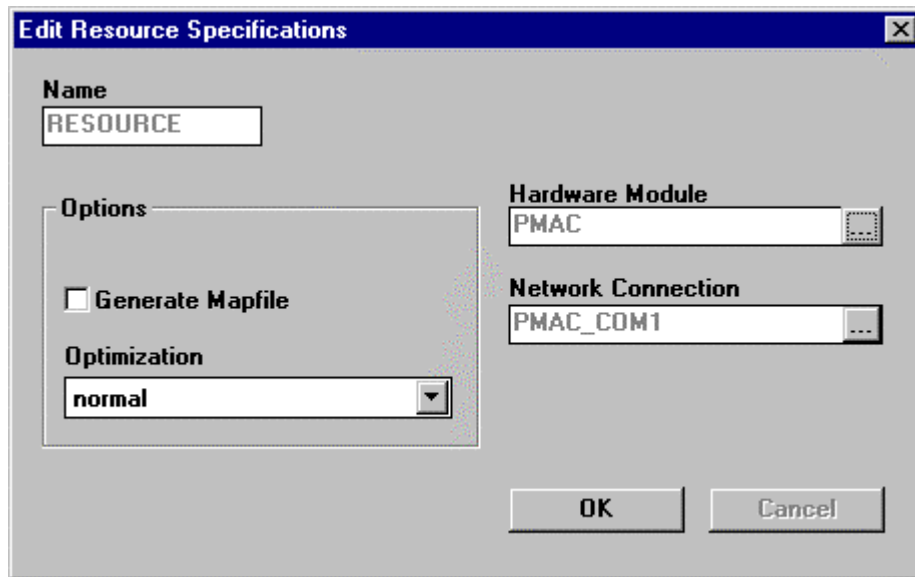


You have just run **LDADD** in **SmartSim32** which simulates the **TURBO Ladder Runtime** firmware on the PC. Play with it for a while and then close the **ControlX Editor**, **Test and Commissioning** and **SmartSim32** Windows.

Go Online to Run & Test Program in TURBO PMAC

Editing a Resource

Resource is an abstract term defined by IEC1131. In general, you can substitute “PLC” or “controller” for “resource.” There are two resources in the **TURBO PMAC LADDER** software program **PMAC**: (Turbo) & **SmartPLC** (the PC). To change the resource to be the **TURBO PMAC** do the following: Highlight **Configuration\Resource**, then right-mouse click to **Properties**. Change the **Hardware Module** to be **PMAC** and the **Network Connection** to be **PMAC_COM1 .. COM4**. Connect the **PMAC RS232** cable from the **TURBO Auxiliary RS232** port to your PC **COMn<1-4>** port.

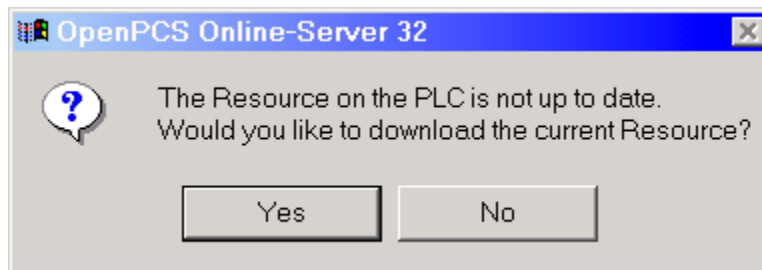


Go Online to PMAC

Select **PLC → Online**. This will start the communication with PMAC and will try to download the PLC to PMAC. If you get the following error message then set **PMAC I variable I44 = 1** using PMAC Executive s/w and retry **PLC → Online**.



On successful communication this will be the message. Select **YES** to download the PLC to PMAC.



Redo similar test as we did in the PC simulation mode. You have now run the minimum program in both a PC and the PMAC.

FUNDAMENTALS OF LADDER, FUNCTION BLOCK AND SFC

Ladder Diagram (LD)

The ladder diagram is a graphical language based on the relay ladder logic – a technique commonly used to program current generation PLCs. However, the IEC Ladder Diagram language also allows the connection of user-defined function blocks and functions and so can be used in a hierarchical design.

The basic principle of Ladder Logic is current flow through networks. Generally, Ladder Logic is restricted to processing Boolean signals (1=True, 0=False).

A **Network** is restricted by so called margin connectors to the left and to the right within the Ladder Editor. The left margin connector has the logical value 1 (current). There are connections that conduct current to elements (variables) that conduct current to the right hand side or isolate depending on their logical state. The result of the procedure depends on the arrangement of elements and the way they are connected (AND = serial; OR = parallel).








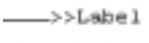
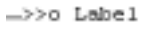



Networks consist of the following graphical objects:

- Connections (horizontal or vertical lines, and soldered points)
- **Contacts, Coils, Control Relays**
- **Function blocks and Functions**
- Jumps (Graphical elements for control flow)

Programming in Ladder Diagram

The programming language Ladder Diagram is suited for logical expressions of binary variables.

Programs in Ladder Diagram are shown with the following graphical elements:



	Contact: Check variable for value TRUE
	Negated contact: check variable for value FALSE
	Coil: store result to variable
	Negated coil: store negated result to variable
	Set: set variable to TRUE if expression evaluates to TRUE
	Reset: set variable to FALSE if expression evaluates to TRUE
	Unconditional jump to label "Label"
	Conditional jump: if result is TRUE, jump to label "Label"
	Negated conditional jump: if result is FALSE, jump to label "Label"
	Unconditional return: terminate current program and return to caller
	Negated conditional return: if result is FALSE, terminate current program and return to caller
	Conditional return: if result is TRUE, terminate current program and return to caller

Each ladder network is divided into two parts. The left part is reading input variables (contacts). The right part is using the evaluated expression and storing the result into other variables (coils), or executing conditional instructions based on the result of the expression.

The terminating right part of a ladder network can be

- An assignment
- A conditional jump to a label
- A conditional return to the caller

The result of an expression can be stored to a variable:

- Store result to variable 
- Store negated result to variable: 

Function Block Diagram (FBD)

This is a graphical language for depicting signal and data flows through function blocks - re-usable software elements. FBD is useful for expressing the interconnection of control system algorithms and logic.

Function blocks should be regarded as the basic building blocks of a control system. The standard provides facilities so that well defined algorithms or control strategies written in any of the IEC languages can be packaged as re-usable software elements.

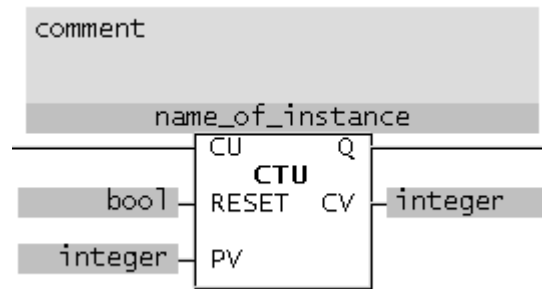
IEC 61131-3 encourages the development of well-structured software that can be designed either from top-down or bottom-up. One of the main features in the standard to support this is the use of function blocks. A function block is part of a control program that is packaged so that it can be re-used in different parts of either the same program, or in a different program or project. A function block may provide the software solution to some small problem, such as creating a pulse timer, or may provide the control to a major piece of plant or machinery, for example, the control of a pressure vessel.

Function blocks may be used for frequently needed functionality. Function blocks may contain several input parameters and several output parameters. A function block keeps up its variable values from one invocation to the next. Blocks may be reused during the next invocation if they are not assigned new values.

Use of Function Blocks

A **Function Block** is represented as a rectangular graphical symbol. The connectors on the left- hand side represent the input variables, while the connectors to the right represent the output variables. Within the rectangle, you see the name of the function block. The names of the input operands are situated to the left, while the names of the output operands are to the right. The instance name is above the rectangle. Calling function blocks is nothing but calling instances. The instances can occur many times in the program.

Example of a function block:



A function block constitutes an autonomous network. Input and output operands that are not used will not be assigned variable identifiers. The corresponding connectors will then remain without labels. If parameters are passed outside the function block, the function block symbol will be drawn without any connectors.

There are two kinds of function blocks. Manufacturer's function blocks and User function blocks. The user function blocks are those created by the user to be used in the program.

Sequential Function Chart (SFC)

This is a graphical language for depicting sequential behavior of a control system. It is used for defining control sequences that are time- and event-driven. This is an extremely effective graphical language for expressing both the high level sequential parts of a control program as well as programming low-level sequences, e.g. to program an interface to a device.

Elements of the Sequential Function Chart

SFC-plans are a tool for formulation of control flow of a technical process, which are characterized by change of states. Every state transition is coupled on certain conditions.

The sequential function chart offers the following language elements:



Step:

A step contains many actions. Actions contain code fragments.

A step that is executing is called "active."

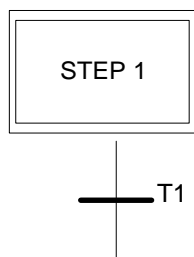
If a step is active, the contained actions will execute.

A step can be activated by:

switching of a previous transition,

a jump element,

Setting the initial flag (c.f. initial step).



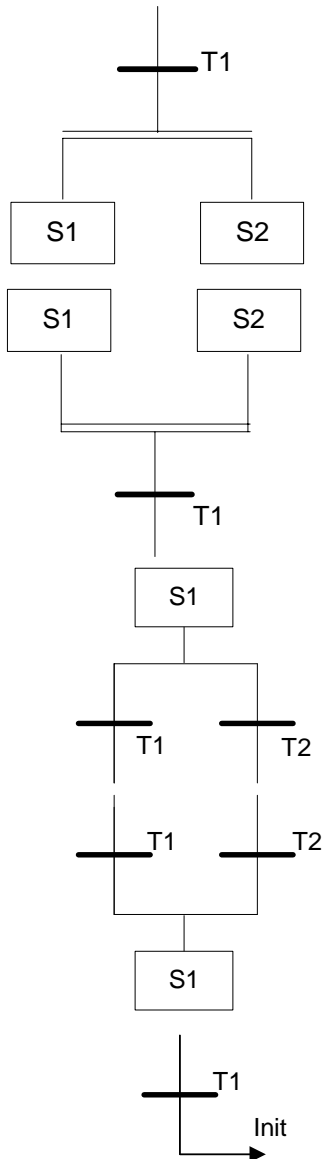
Initial step:

Initial steps are active at the beginning of the program.

Positions in the plan could be marked by initial steps, at which the execution starts on program start.

Transition:

Switching of transitions controls the program flow temporally and structurally. A transition will switch if the transition condition is true and all previous steps are active. Once the transition switches, all previous steps become inactive and all following steps become active.



Simultaneous sequences:

One transition may set active multiple steps at the same time, starting a parallel chain.

If all previous steps of transition T1 are active and the transition condition is TRUE, all following steps (e.g., S1, S2) of the simultaneous sequence will activate.

Convergence of simultaneous sequences:

The chains of a simultaneous sequence are converged into a single transition.

If all previous steps (e.g., S1, S2) are active and the condition of the following transition (T1) is TRUE, all previous steps will be deactivated while the steps following transition T1 will be activated.

Divergence of sequence selection:

Selection of a sequence step chain.

If the step before the divergence is active, all transitions (e.g., T1, T2) are checked from left to right. The first transition evaluating to TRUE will switch, deactivating the step before and activating the step after.

Convergence of sequence selection:

The chains of a divergence of sequence selection are converged into a step.

If one of the transitions switches, the steps before it will be deactivated, and the Step following it will be activated.

Jump:

The program flow is continued at another location.

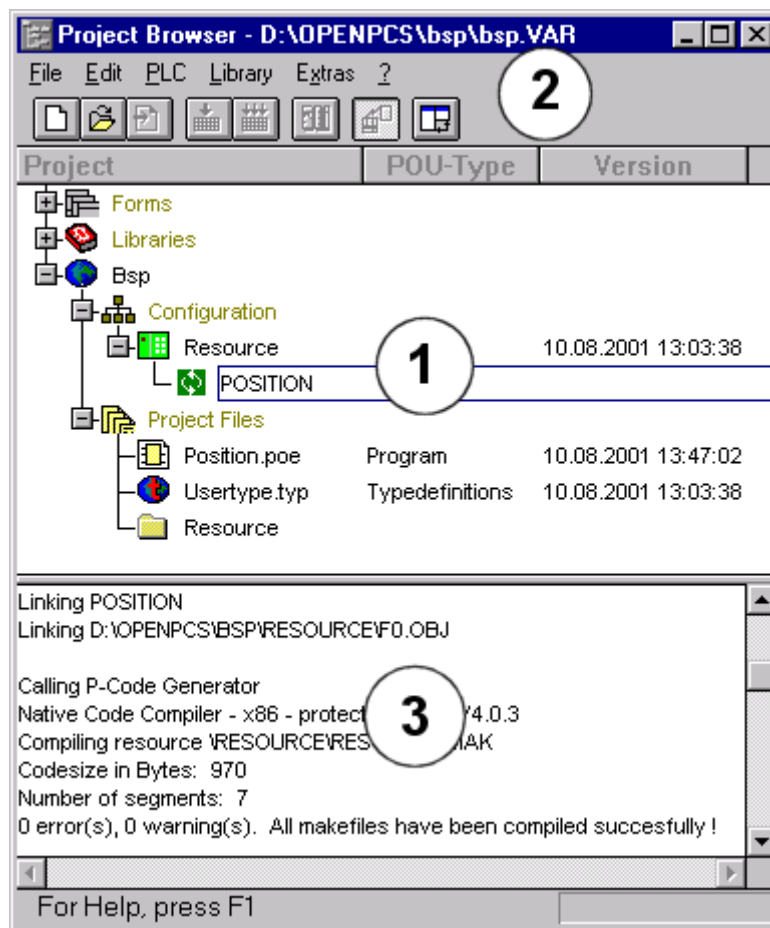
The name of the jump is the name of the active step if the previous transition (T1) switches.

THE PROJECT BROWSER

The tool **Project Browser** is used for the management of projects and the files belonging to projects. With the Project Browser you can structure your work into projects. A great advantage of OpenPCS is the reuse of blocks in other projects. Please note the following hints:

- Avoid creating several OpenPCS-projects for related work. One machine, one network of PMACs, or even one plant might be one project.
- Use separate projects for unrelated work; this increases your overview and prevents you from mistakenly modifying wrong code.









The project management window consists of three parts:



1. **The project tree.** Here you can see all information concerning the project. The column headed POU-Type informs you about the use of the program organization unit. The column Version shows the last saving of a pou-file.

The width of the columns can be arranged individually and will be saved while closing the Project Browser.

2. **The menu panel and the toolbar.**
The toolbar:

- | | |
|---|---------------------------------------|
|  | Create new project |
|  | Open an existing project |
|  | Add a file to the project |
|  | Generate executable code |
|  | Generate all new |
|  | Edit resource |
|
 | |
|  | Go online |
|  | Separate the desktop for applications |

3. **The output window.** Output of the compilation or the online operation are shown here.
In the following chapters the functionality of the project browser management is explained in detail.


Start of the Project Browser

Start **OpenPCS** from the start menu: Select **Start --> Programs --> OpenPCS --> OpenPCS**.

Managing projects

Create a new project

This can be done in two ways:

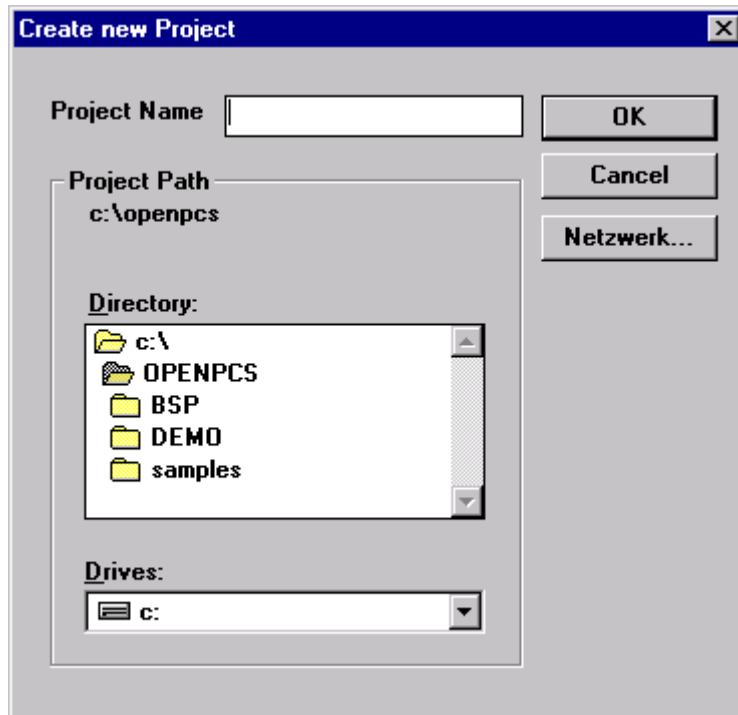
- By the toolbar: Click on the button **New project:** 
- By the menu: Click in the File menu on **Project --> New...** .

Notes

The project name has a maximum length of eight characters. If you try to enter a name with more characters, the cursor will jump to the initial position again.


You can create the project in any directory, but it is recommended to hold the projects in your OpenPCS-directory.

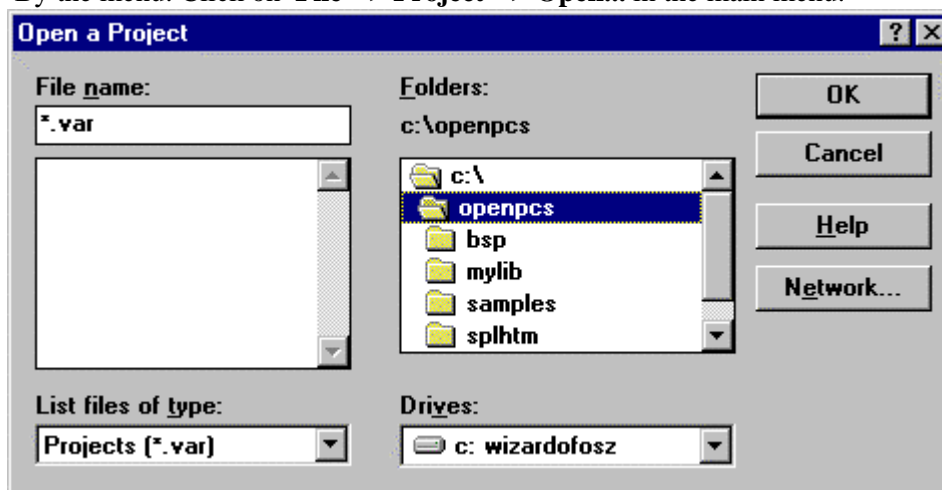
A subdirectory, which has the same name as your project, will be created automatically. This directory contains all files that belong to your project.



Open an existing project

There are three ways to open a project:

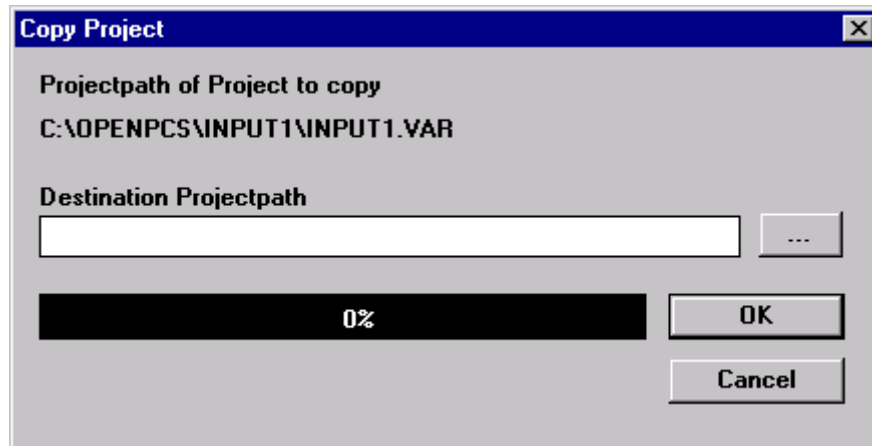
1. In the **File**-menu: Here you find under the item **Recent Projects** a list of the last opened projects; the file, which you are looking for, could be contained in this list.
2. By the toolbar: Click on the button **Open Project** 
3. By the menu: Click on **File --> Project --> Open...** in the main menu.



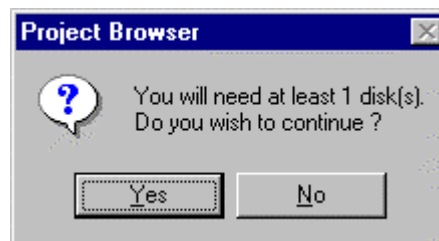
--> Select the desired project in the dialogbox or look for it in the folders. The project files have the suffix **.var**.

Copy a project

To do this, select the menu item **File --> Project --> Copy...** in the project browser. Then you get the following dialog box:

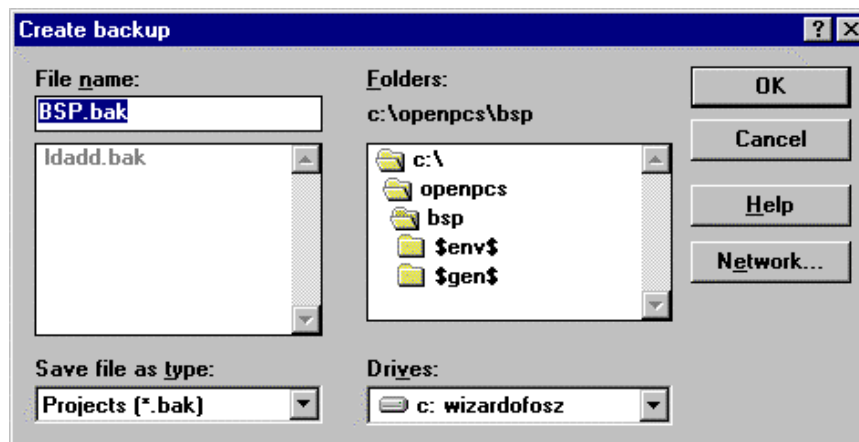


Enter the destination path, and start with the button **OK**. If you want to copy a project to floppy disk the project browser informs you how many diskettes you will need.



Create a backup copy

Select the menu item **File --> Project --> Backup...** to create a safe copy of the current project. The following window appears:



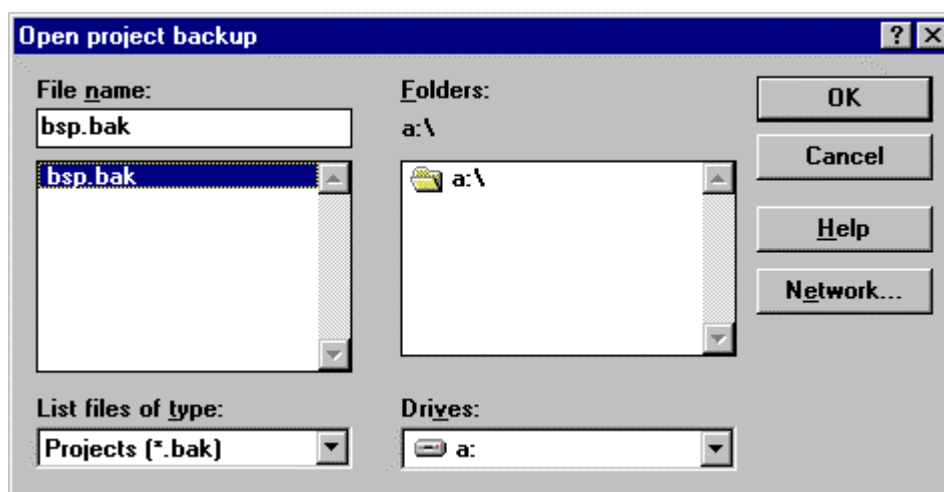
1. Enter the name of your safe copy in the field **File name**, or select an existing safe copy from the list that will be overwritten then.
2. Select the folder in which the file will be created.
3. Accept your entry by the **OK**- button.

The safe copy will be created.

Restore a project

If you created a safe copy, you can restore your project.

1. Select the menu item **File --> Project--> Restore...**
2. Enter the name of your safe copy into the field **File name**, or select a name from the list below:



3. Accept with the **OK**- button.

A dialog box will be opened in which you can select the target directory:

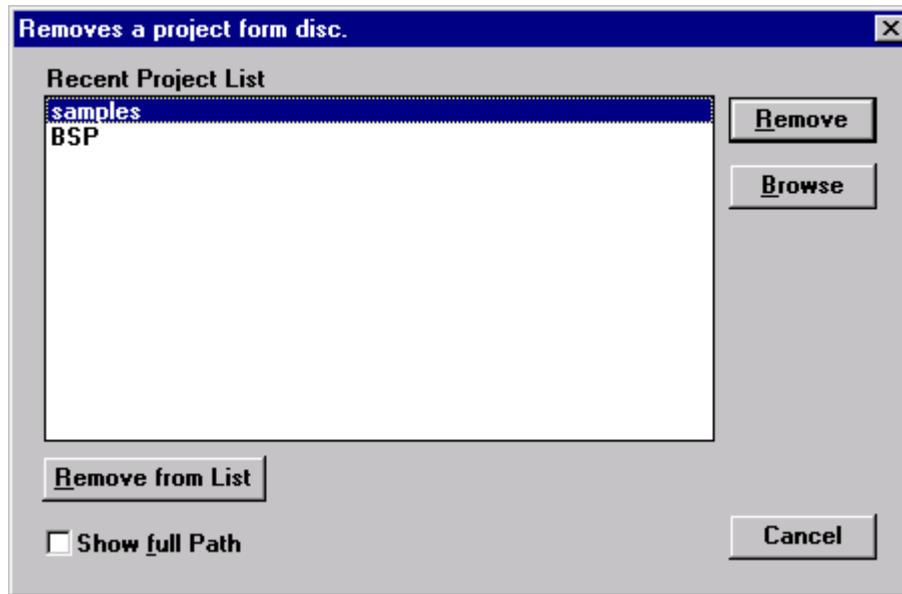


4. Select a project directory in which you want to restore your project.
5. Accept your selection with the **OK**- button.

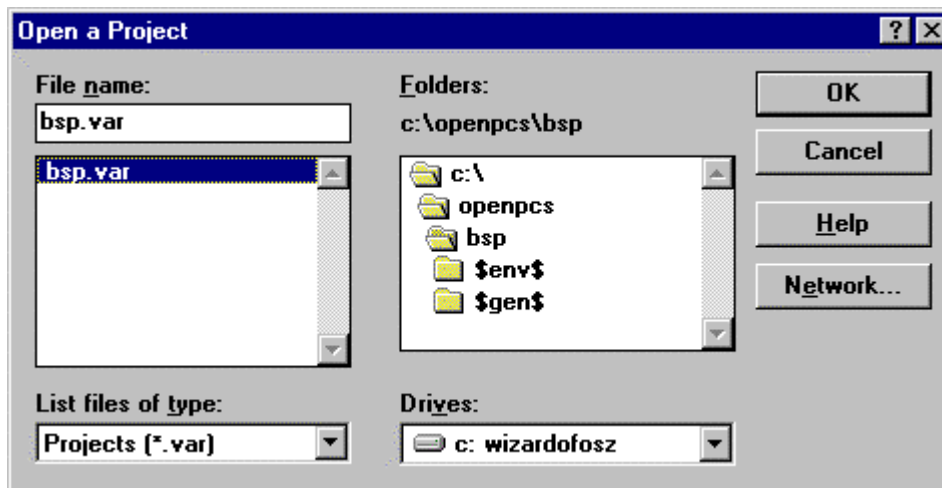
The project is restored. Messages will be displayed in the output part of the project browser window.

Deleting projects

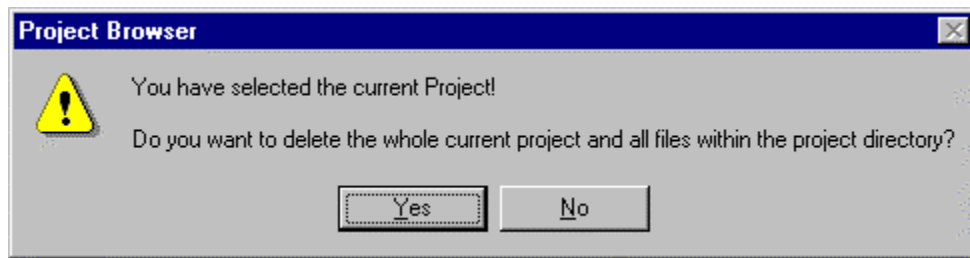
You can delete the current project or every other saved project. Just select the menu item **File --> Project --> Delete** and the following dialog appears:



Now you can select a project out of the **Recent Project List** or use the **Browse-** button to search a project.

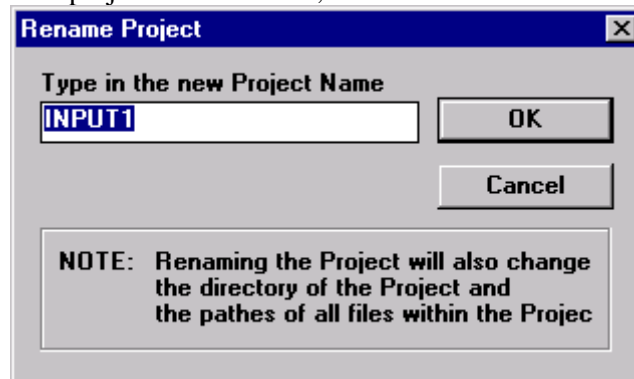


Chose a directory and select in the left column the file you want to delete. Clicking the **OK** button starts the process of erasing after you have answered a security question.



Rename project

If you want to give the current project another name, select the menu item **File --> Project --> Rename...**




Notes

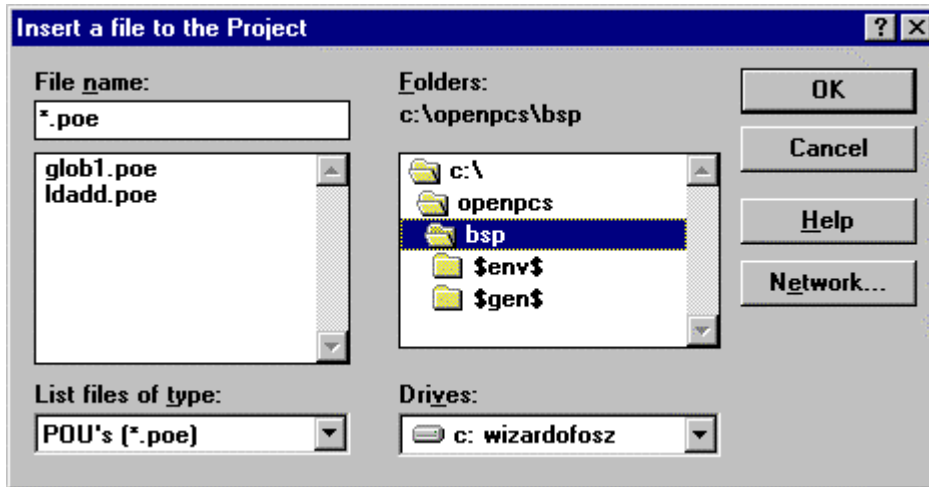
The project name has a maximum length of eight characters. If you try to enter a name with more characters, the cursor will jump to the initial position again.

You can create the project in any directory, but it is recommended to hold the projects in your OpenPCS-directory.

A subdirectory with the same name as your project will be created automatically. This directory contains all files that belong to your project.

Add Files

You can add at any time more files to an existing project. Click on the item **Add File** in the menu Project or click on the icon **Add file**  and the following dialog box will be opened. Note that you must have the **Project Files** part of the Project Tree selected:



Not only can you write files with the editors of **OpenPCS** (LDD & SFC), you can also import type definition and type declaration files as well as resources. Furthermore, you can register files in one project, even if they were created by other programs, for example by:

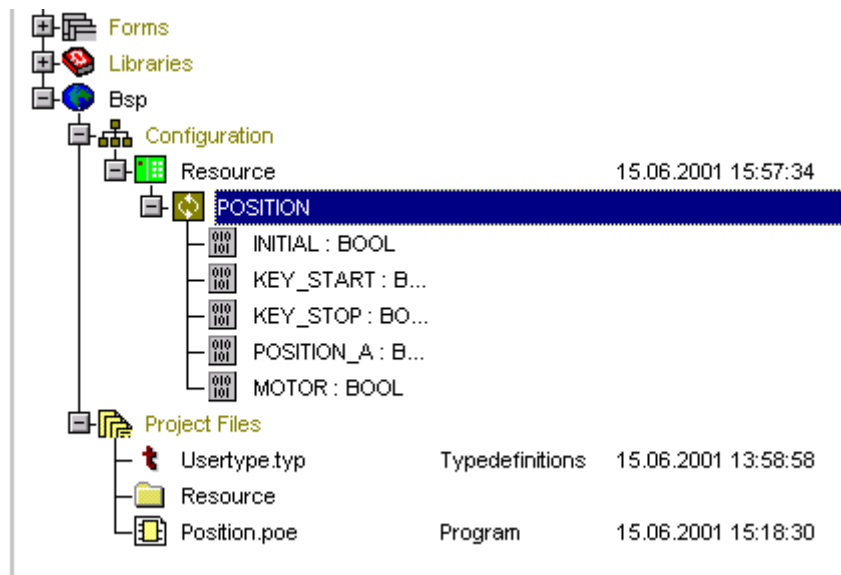
- Microsoft Word
- Microsoft Project
- Microsoft Excel
- AutoCAD

Select the desired file type in the popup menu and open the corresponding directory. There you can select the file you want to add. A multiple selection is possible by holding down the **shift** or **Ctrl** key while clicking the mouse. These files will be copied in the current directory of the browser and can be opened for editing by a double click.

The Project Tree

Basics of the Project Tree

The project tree is generated automatically for the project.



Note:

Only the variables of compiled resources can be displayed.

The individual branches of the project tree will be described in the following chapters.

The Project Directory

A project (in this case **Bsp**) consists of two subdirectories:

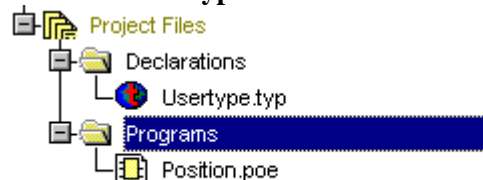


These subdirectories are the same for all projects, and are generated automatically together with the project.

The resources of the project are in the branch **Configuration**. We examine the subject **Resources** more closely in the chapter **Resources**.



The files, e.g. LDDs, SFCs, variable- or type-declarations, can be found under **Project files**. We examine these in the chapters **Creation of modules** and **Type definitions**.



Libraries

The Project Browser features functions required to use libraries, so you can re-use commonly used functionality by putting functions and function blocks into a library of POU's.

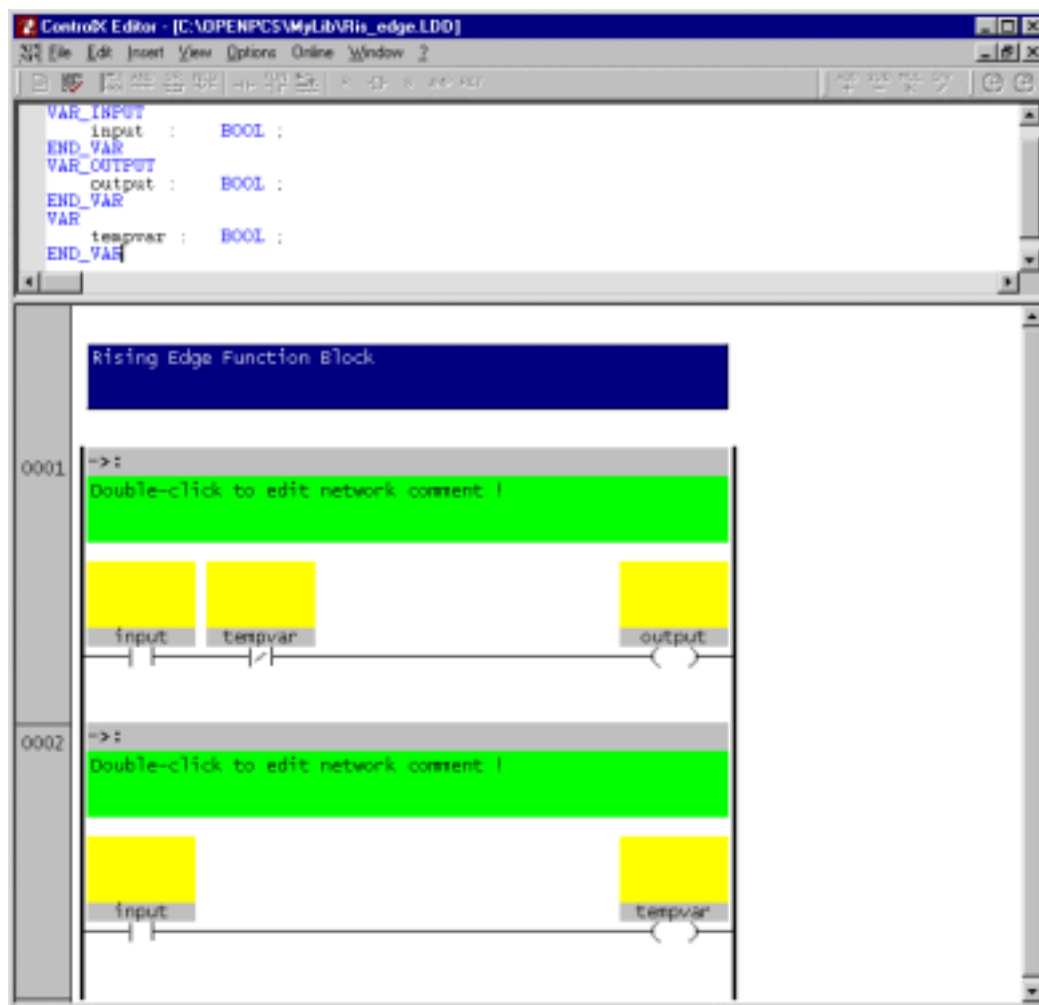
To create a library, proceed as if creating any normal OpenPCS project. Be sure to perform a syntax check when finished creating POU's (functions or function blocks) in your library project.

Note:

If you receive a library from your supplier, you will not have to create that library. Proceed with installing this library instead!

Example:

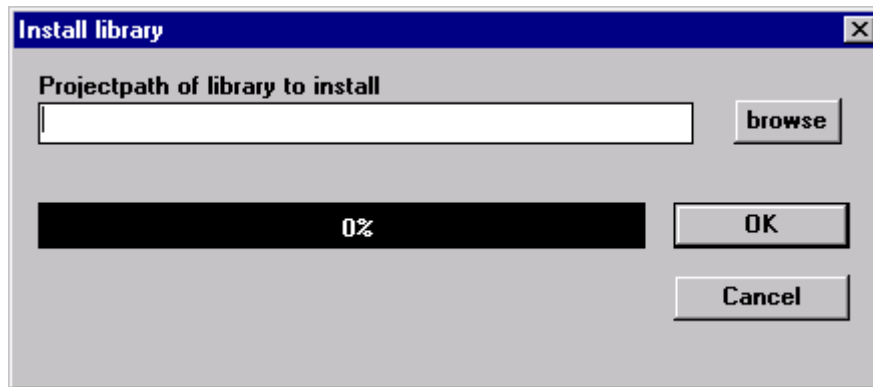
1. Start the Browser and create a new project named **MyLib** using **File --> Project-->New...**
2. Create a function block named **Ris_edge** (for edge detection): **File --> New --> Functionblock --> LD**. Implement this function block as shown below:



3. Invoke a syntax check in the POU editor with **File-->Syntaxcheck**.

Installing a library

Before you can use a library, you have to install it on your PC. Use **Library-->Install** within the Browser:



Use the **browse-** button to locate the .VAR file representing your project. If you created the library yourself, this will be in the directory you specified when creating the library project with **File --> Project-->New....** If you received the library on a disk, this can be something beginning with **A:**. During installation, the library project will be copied into a sub-directory of **<windows>\openpcs.4xx\Lib**.

Example:

1. Create a new project in the Browser using **File --> Project-->New....** . Name that new project **TEST**.
2. Select **Library--> Install**.
3. Now use the browse-button to locate the MyLib-project you created just before and press **Install**.

Adding a library to a project

After installation, all files needed for the library will be present on your computer. But the functions and function blocks in that library will not be automatically available in your projects. You have to **add** the library to the project first, using **Library-->Add**.

Example:

- Select **Library-->Add** and in the dialog shown find your library **MyLib**, then press **OK**.

Un-Install a library

If you want to get rid of a library installed on your PC, select **Library-->Uninstall**. In the dialog shown, select the library to get rid of and press **OK**.

Example:

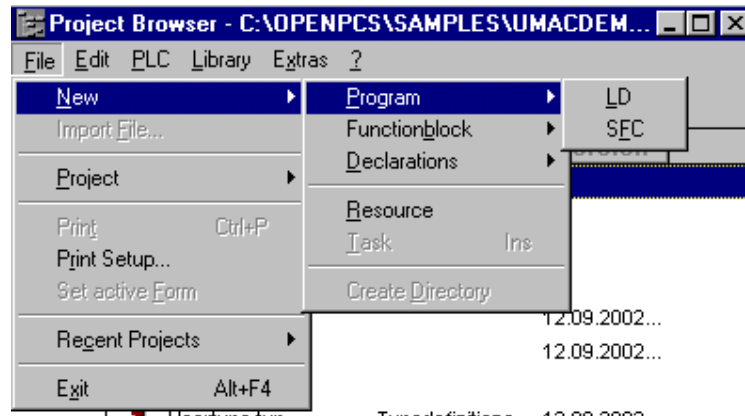
- Select **Library-->Uninstall**. In the dialog, select **<Windows>\openpcs.4xx\MyLib**.
- Press **OK**, and **MyLib** is no longer available as a library.

Work with Modules and Type Declaration Files

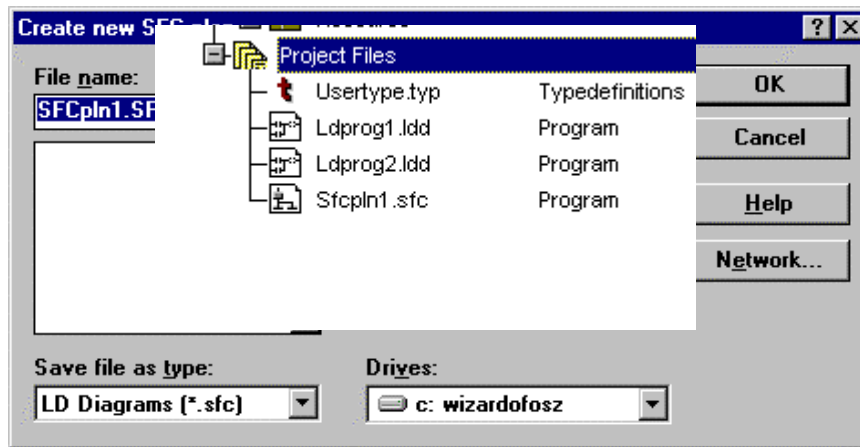
Creating Modules

The different types of program organization unit (POU) files, SFC-files (SFC = Sequential Function Chart) and LD-files (LD = Ladder) can be created by the project browser as follows:

Select **File-->New-->** from the menu, and decide if you want to create a program, a function or a function block. After that you can select the desired type:



Enter in the next dialog the name of your module:



The new module is added to the project tree under **Project files**:

Editing Modules

Modules can be edited any of three ways:

1. Double click the module
2. Right click on the module and select **Open** from the context menu
3. Mark the module and select the menu item **Edit --> Edit**.

The corresponding editor will be opened according to the type of the file. You can graphically create programs with the SFC-editor. See the chapters on the respective editors for more details on how to edit files.

Copy

The various files of the modules and type declaration files can be copied within a project. For this purpose mark the chosen POU-, SFC-, LD- or type declaration files and

- click the item **Copy...** in the **Edit** menu, or
- click the right mouse button on the selected file and choose **Copy** from the context menu.

The following dialog box will be opened:



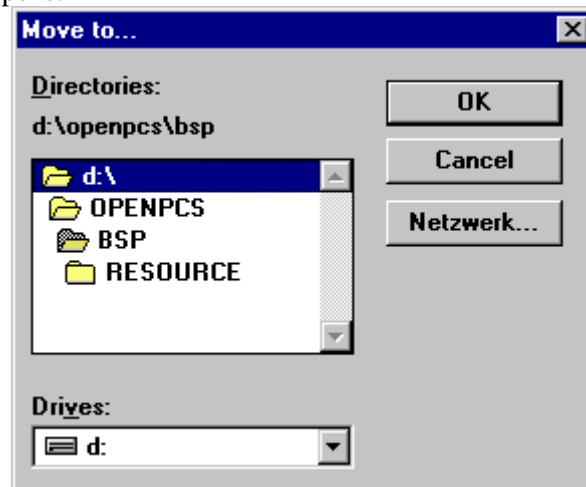
Select a target directory and name the file you want to copy. Be aware that this name must not be identical with the original name of the file. If name or path are incorrect you will get an appropriate message and the dialog box will open up again.

Move

To move modules, type declaration files within a project, mark the selected files and

- click the item **Move** in the **Edit** menu, or
- click the right mouse button on the selected file and choose **Move** from the context menu.

The following dialog box opens:



Select a target directory and name the file you want to copy and press OK.

Global Resource Variables

Variables are names that are used to represent memory locations holding data. Variables can be classified as follows:

- Variables that are assigned to physical input- or output-locations, or PLC-memories.
- Variables that are only used internally to store intermediate results.

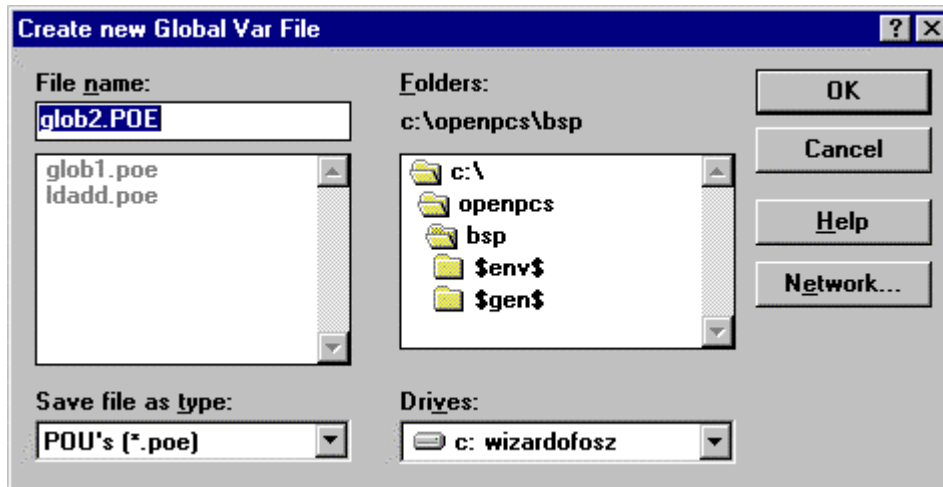
Global resource variables are accessible from all locations of the resource.

In **OpenPCS**, there are two kinds of global resource variables:

- Global variables. These are variables without hardware addresses, e.g. for intermediate results.
- Direct global variables. These are variables with direct hardware-addresses together with the IO-declarations. These represent the interface to the hardware.

Creation of [direct] global variables

Select the menu item **File --> New --> Declaration --> [Direct] Global Variables**



Enter the name of the variable declaration, and accept with the **OK**- button. The new declaration appears in the branch **Project files** of the project tree:



Editing of [direct] global variables

You have three possibilities to edit [direct] global variables:

- Double click on the declaration in the branch **Project files**.
- Right click on the declaration, and select **Open** from the context menu.
- Mark the declaration, and select the menu item **Edit --> Edit**.

The POU-editor will be opened, and you can declare your variables. Only variables of type GLOBAL can be declared.

Type definitions

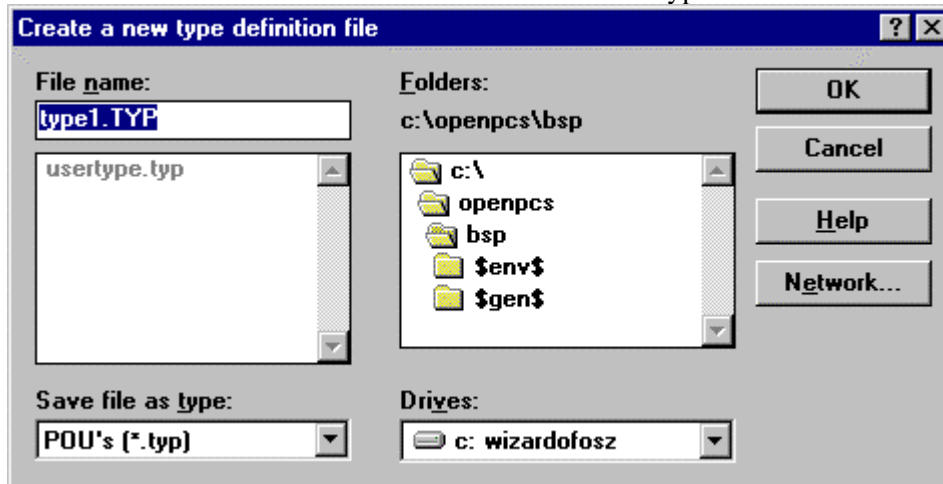
Data types can be either elementary or derived. Elementary data types are defined either in the IEC 1131 or by the manufacturer of the control system. On the other hand, derived data types are defined by the programmer or by the manufacturer.

By creation of the project, a standard type definition **Usertype.typ** will be generated. This one cannot be deleted and is assigned to all resources:

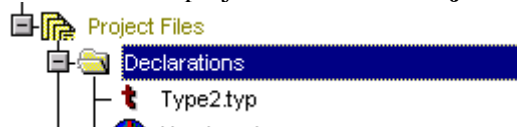


Creation of new type definitions

Select a menu item `File --> New --> Declarations --> Type Definition` :



Enter the name of the type definition and accept with the **OK**- button.
The new type definition will be located in the project tree under **Project files**:



Editing of type definitions

There are three ways to start editing a type definition:

1. Double click on the definition in the branch **Project files**.
2. Right click on the definition, and select **Open** from the context menu.
3. Mark the definition, and select the menu item **Edit --> Edit**.

The POU-editor will be opened, and you can declare your types.

Resources

If a POU was created, OpenPCS defines a resource in order to execute it on your control system. In general, a resource is equivalent to a PLC or a micro controller.

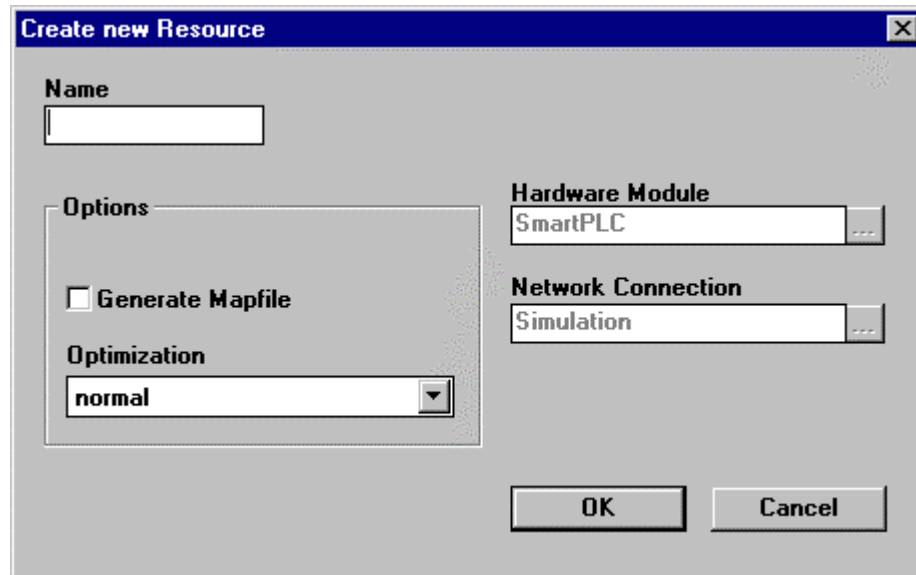
A resource definition consists of the following:

1. A name for identification.

2. The hardware description: Information about the properties of your PLC which will be used by OpenPCS.
3. Information about the kind of communication between OpenPCS and the control system.
4. A list of tasks that are to be run on the control system.

Creation of a Resource

While creating a new project, **OpenPCS** installs a suitable resource that is already defined. If you want to create additional resources click in the menu File on **New --> Resource**. A dialog box appears in which you can define the properties of the resource.




- You can select a resource name with a maximum size of eight characters.
- In this version, the default-Hardware type is the **SmartPLC**, and the other is **PMAC**.
- The network connection is pre-selected as **Simulation** to work with the PLC-simulation of **OpenPCS**. Of course it is possible to change the network connection by clicking the button beside the textfield below **Network connection**.
- After you applied a name for the resource you can select a network connection by clicking the button beside textfield below **Network Connection**. To work with the PLC-simulation of **OpenPCS** (SmartSIM32) select **Simulation**. To work with a **PMAC** you have **PMAC_COM1 to COM4**.

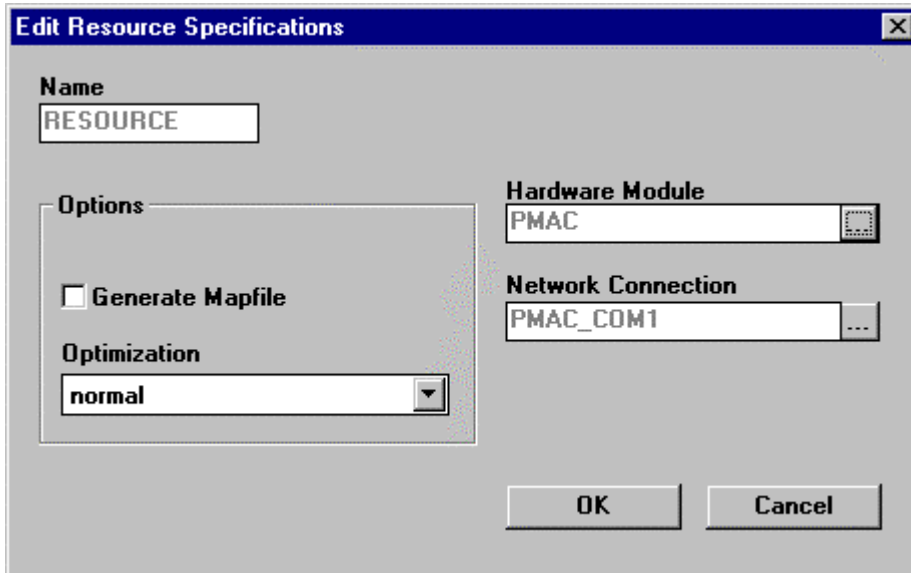
Under **Options** you can:

- select the function **Generate Mapfile**: after generating the code three text files will be created in which you find linker information. These files will be saved in the resource directory named **Pcedata.txt**, **PceVars.txt** and **PceSegs.txt**.
- Under **Optimization**: **Normal** and **Speed Only** are the same and the PLCs are generated in the **Native Code** (PLC execution will be fastest). The **Size Only** mode is mainly used for Debug mode. The Execution will be slower than Native code mode.

Edit a Resource

Before editing a resource, set it as **active** (marked green in the project tree). In general, all possible operations relate to the active resource.

To edit a resource, right click it (if it is not marked as **active** you can do that now) and select **Properties** from the context menu, or mark the resource and click in the toolbar on the icon **Edit resource**: 



The dialog box titled "Edit Resource Specifications" contains the following fields:

- Name:** A text box containing the word "RESOURCE".
- Options:** A group box containing:
 - ☐ **Generate Mapfile**
 - Optimization:** A dropdown menu currently set to "normal".
- Hardware Module:** A text box containing "PMAC".
- Network Connection:** A text box containing "PMAC_COM1".
- Buttons:** "OK" and "Cancel" buttons at the bottom right.

The dialog **Edit Resource Specifications** will be opened:

Adding a Task

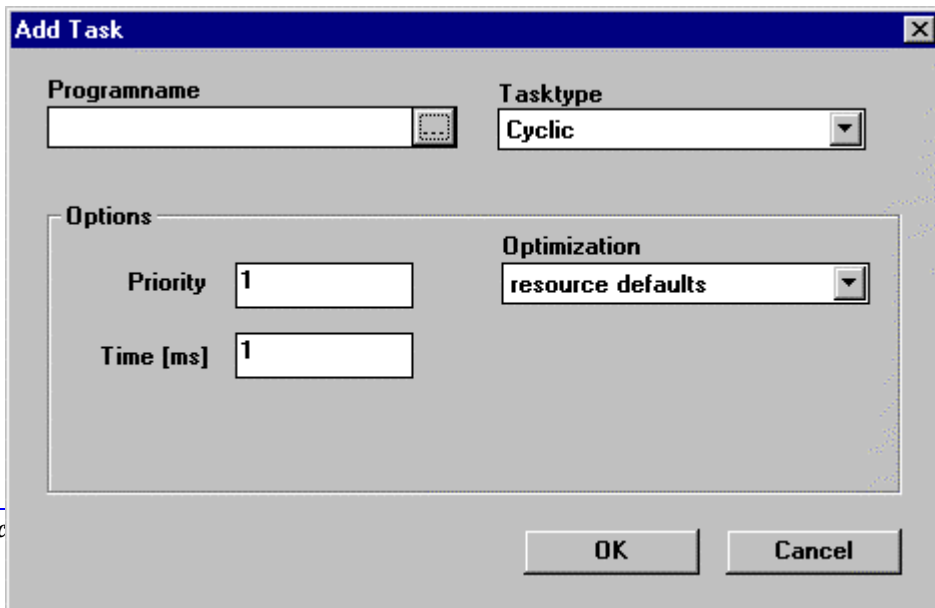
In general, a task is equivalent to a program plus the information how the program can be executed. The definition of a task consists of:

- The name
- The information about the execution of the task
- A POU of type PROGRAM which should be executed in this task

You can add a task in different ways:

- By Drag and Drop: Click on the block and hold pressed the mouse button, and put it to the resource
- Mark the resource, and select the menu item **File --> New --> Task**
- Right click on the resource, and select context menu item **New task**

After adding the task, the task-properties-window will be opened:



The dialog box titled "Add Task" contains the following fields:

- Programname:** A text box.
- Tasktype:** A dropdown menu currently set to "Cyclic".
- Options:** A group box containing:
 - Priority:** A text box containing "1".
 - Time [ms]:** A text box containing "1".
 - Optimization:** A dropdown menu currently set to "resource defaults".
- Buttons:** "OK" and "Cancel" buttons at the bottom right.

Note:

The task name depends on the program name, and can't be changed. To complete the task definition, you must specify the information, how the task can be executed:

Tasktype:

- Cyclic
- Timer controlled

Cyclic tasks will be executed when no timer tasks are ready to run. The priority that can be specified in the task properties will be interpreted as a cycle interleave, e.g. priority = 3 will have this task executed only every third cycle. No particular execution order is defined by OpenPCS amongst multiple cyclic tasks.

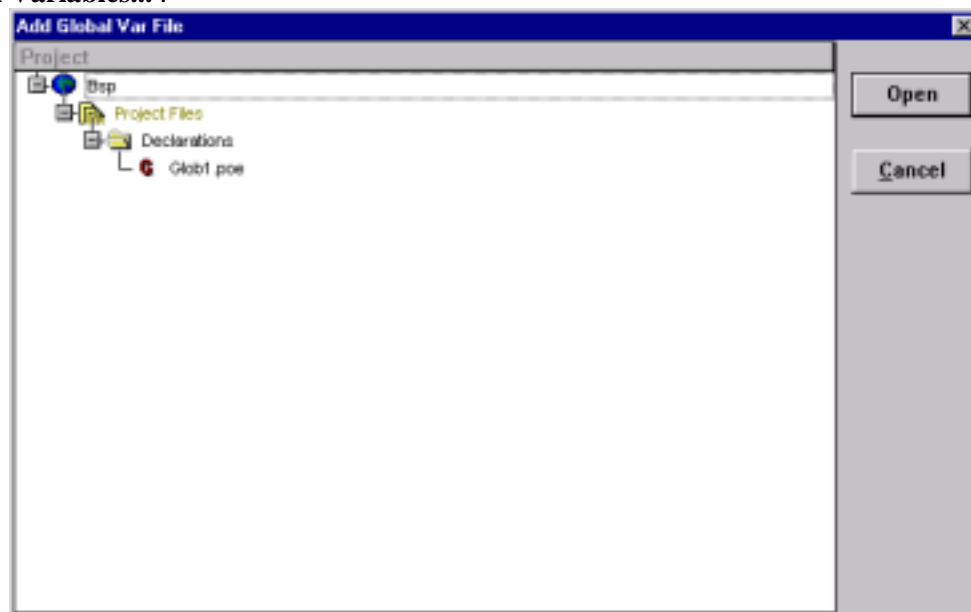
Timer tasks will be executed every N milliseconds, with N specified in the task Time properties.

Under **Optimization: Normal and Speed Only** are the same and the PLC's are generated in the **Native Code** (PLC execution will be fastest). The **Size Only** mode is mainly used for Debug mode. The Execution will be slower than Native code mode.

Adding [direct] Global Variables

Global variables, which are added to a resource by this form, are global for resources. These can be added in either of two ways:

- **Drag and Drop:** Click on the icon of the [direct] global variable definition, hold pressed the mouse button, and put it to the resource to which it should be added.
- **From the Edit-menu:** Mark the resource, and select the menu item **Edit --> Link to --> [Direct] Global Variables...**

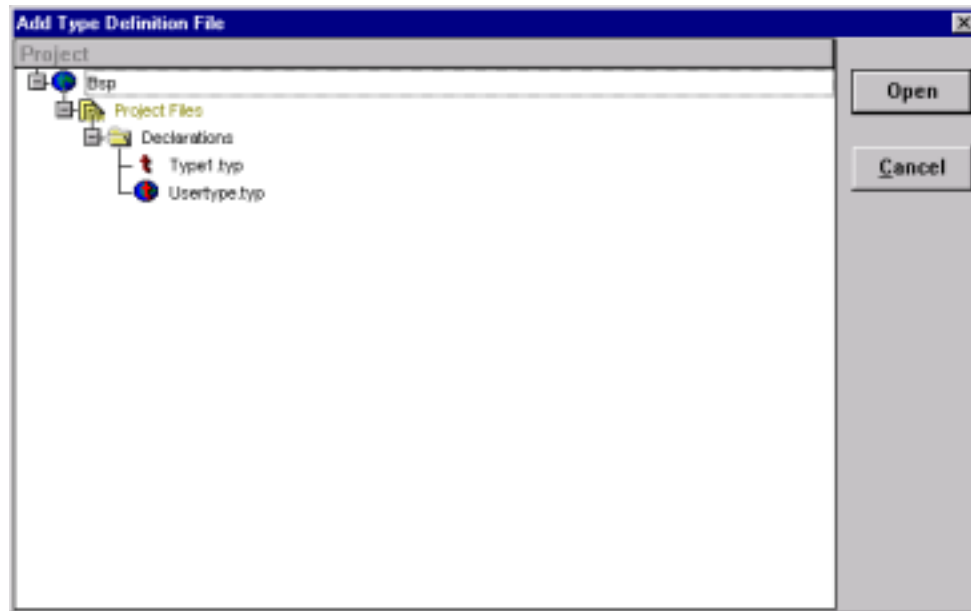


Select the file which you will assign to the resource, and accept this with the button **Open**.

Adding Type Definitions

Type definitions are added as follows:



- **Per Drag and Drop:** Click on the icon of the type definition, hold pressed the mouse button, and put it to the resource to which it should be added.
- **By the Edit-menu:** Mark the resource, and select the menu item **Edit --> Link to --> Typedef...**



Mark the type definition, which you want to add to the resource, and accept by **Open** -button.

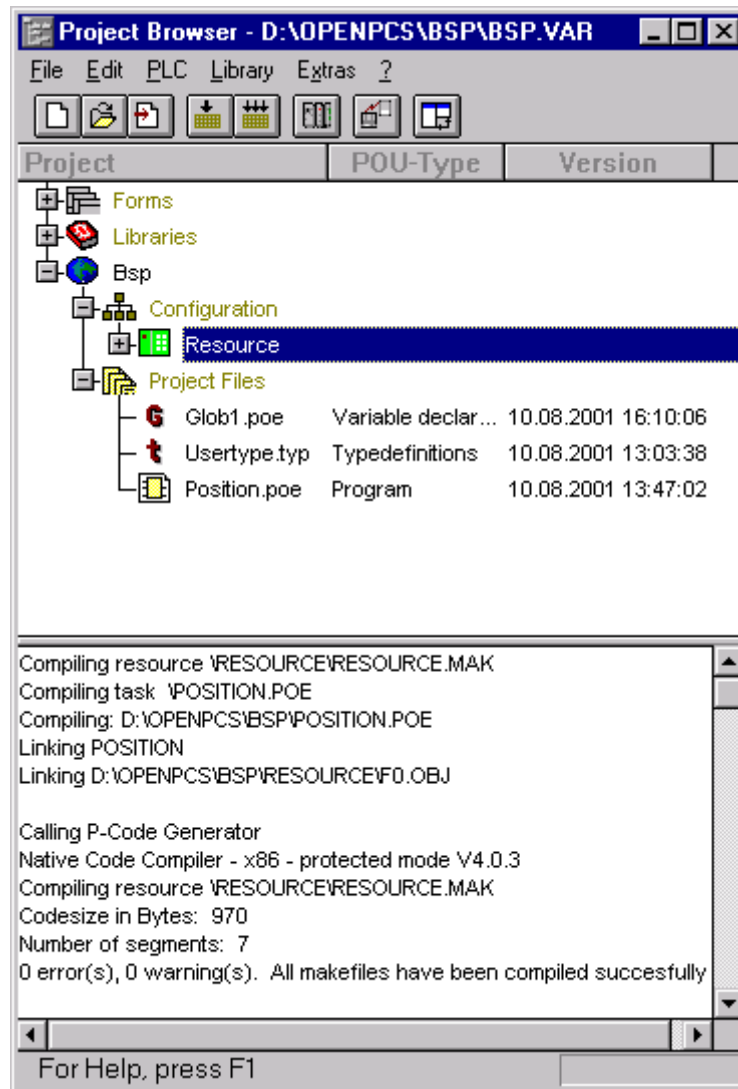
Generate Executable Code

You have three possibilities to generate executable code of a resource:

- Mark the resource, and click on the icon **Generate code**  in the toolbar.
- Mark the resource, and select the menu item **PLC --> Build [All]**.
- Right click on the resource, and select **Build [All]** from the context menu or
- click the icon in the toolbar for **Build [All]** 

The menu item **Build All** will recompile even unmodified portions of your application.

The compiler messages can be read in the lower window of the project browser.



If errors occur, they will be displayed:

```
Compiling resource \BSP_RES\BSP_RES.MAK
Compiling task \POSITION.POE
Compiling: C:\OPENPCS\BSP\POSITION.POE
C:\OPENPCS\BSP\POSITION.POE(3,2,5) : error A3027 : Variable/Name not declared!
All in all 1 errors occurred while compiling !
```

When you double click on an error message, the corresponding editor opens, and the cursor is placed at the line at which the error occurred.

Test and Commissioning

To check a program, you need a PLC. You can use the TURBO PMAC or the PC simulator (the SmartSIM32) that is distributed with **OpenPCS**.

Going Online

Going Online can be done as follows:
Highlight **Configuration/Resource**

- Double mouse click it.
- Right mouse click **Open**.
- Select menu **PLC->Online**.

Starting and Stopping the Program

Press one of the buttons from Test and Commission 32 window.



PLC →STOP

Select **PLC →Stop** to immediately stop the program



PLC →Coldstart

Select **PLC →Coldstart** to perform a cold start. All variables will be initialized to the initial value as programmed.



PLC →Warmstart

Select **PLC →Warmstart** to initialize all normal variables, but keep the values of all variables programmed as RETAIN.



PLC →Hotstart

Select **PLC →Hotstart** to continue execution where it stopped, not initializing any variable.

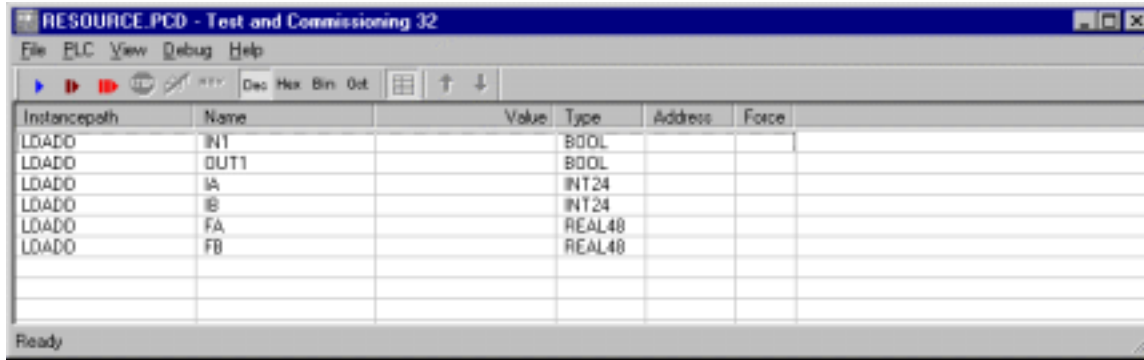
Mouse click the **Blue Arrow** to cold start the program. You should see **IN1** and **OUT1** go to FALSE and the other variables go to zero.

Watching Variables


Open the tree branches under the Program Task being run on the resource. You will see a list of variables for that task that can be watched. They can be watched as follows:

- Double click on the variable that you want to watch.
- Right click on the variable, and select **Watch variable** from the context menu.
- Mark the variable, and select the menu item **PLC --> Watch variable**.

The variables should appear in the **Test and commissioning** -window with instance path, type, value, and status are displayed. These variables are permanently updated during the program execution on the PLC.



To remove variables from the list you have three possibilities. Mark the variable by clicking it with the left mouse button, and then –

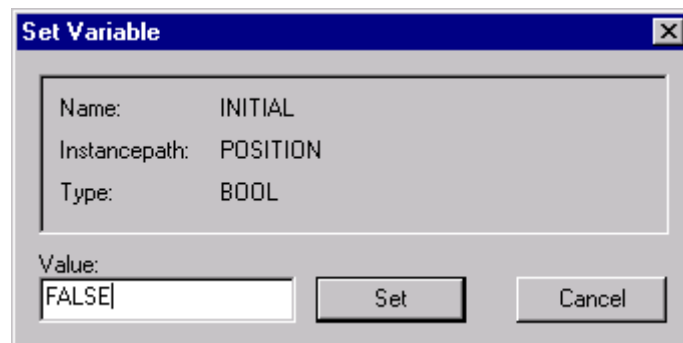
- click on the corresponding symbol in the toolbar .
- use the **del**-key
- or select the item **Remove Variable** in the menu **Debug**.

Set Variables

To influence the behaviour of your control program for test cases, you can set variables to specific values in either of two ways:

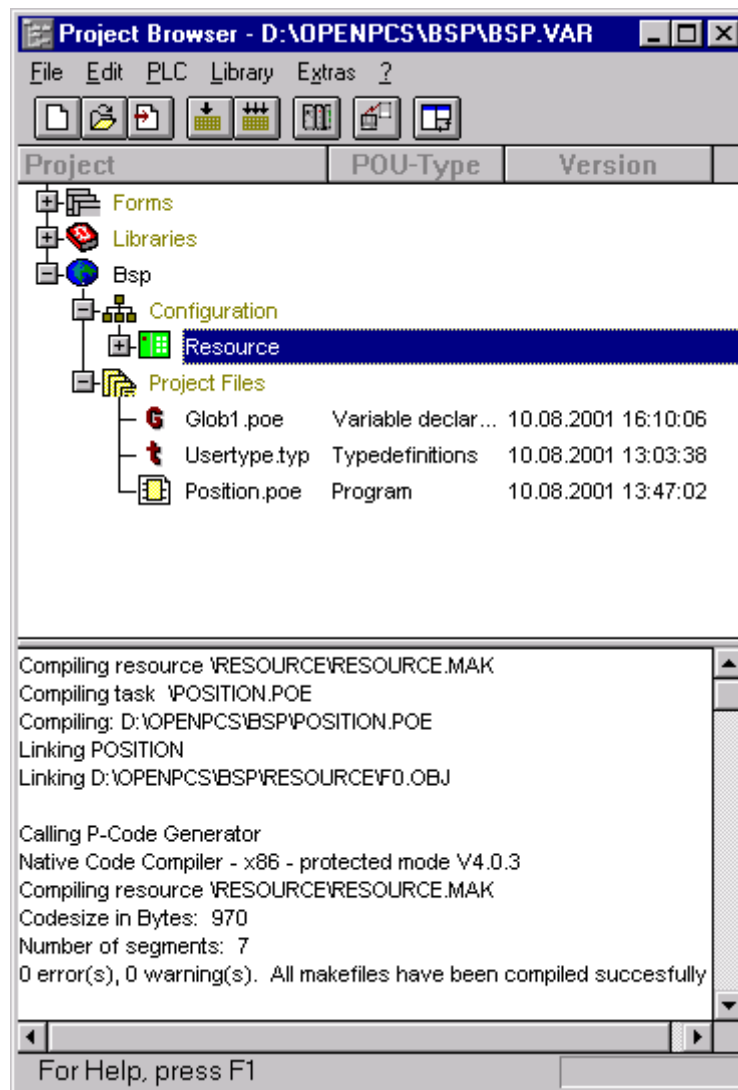
- Mark the variable in the T+C, and select the menu item **Debug --> Set variable**.
- Double click directly on the variable in the T+C.

Enter the new value and accept by **Set**-button.



The Online-Editor

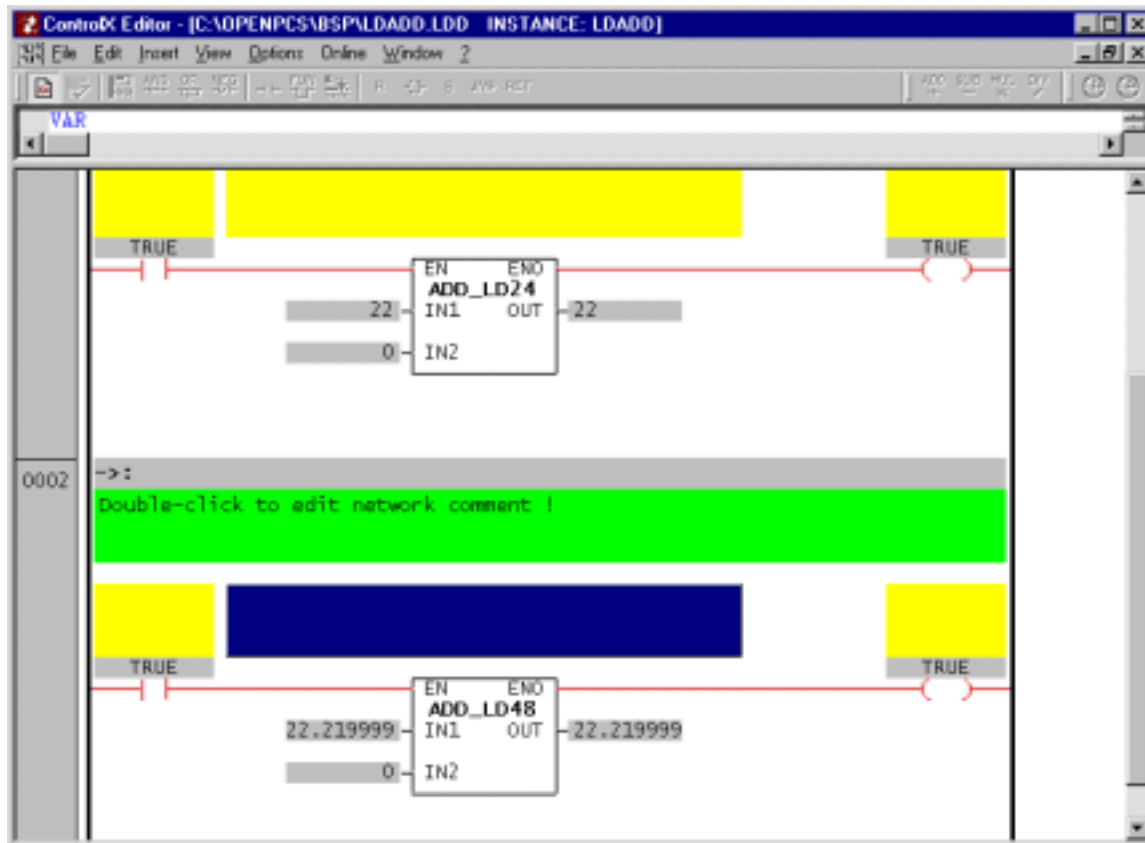
With the online-editor you can watch entire programs or function blocks. The online-editor is only available if you are online. The program or function block that you want to watch must belong to the resource that you loaded into the PLC.



You have three ways to start the online-editor for a program:

- Mark the instance of the POU that you want to watch, in the branch under the corresponding resource, and select the menu item **PLC --> Online Editor**.
- Right click on the instance and select 'Open' from the context menu.
- Double click the instance of the POU (i.g. the task)

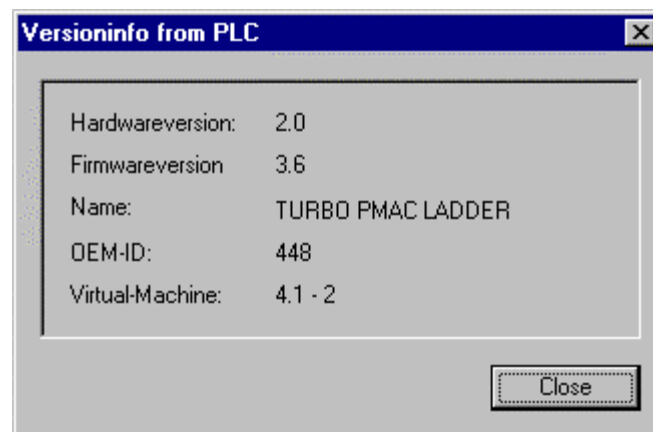
The online-editor will be opened.



Hardware Info

This menu is available only in the online-mode.

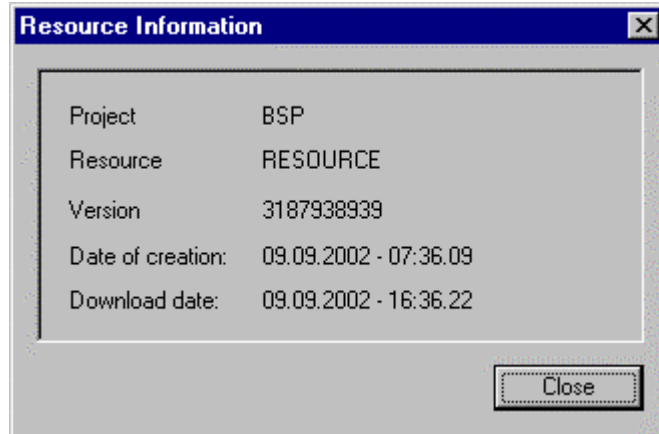
Mark the active resource and select the menu item **PLC --> Hardware Info** in the project browser.



Resource Info

This menu is available only in the online-mode. The project name, the resource name, and the version number (which is internally created and assigned to a specific compilation) are displayed.

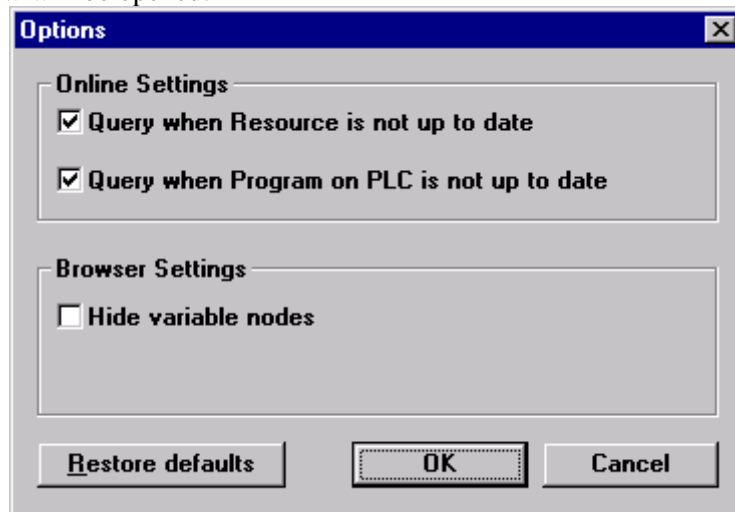
You can display the resource info by marking the resource and selecting the menu item **PLC --> Resource Info** in the project browser.



Configuration of the Project Manager

Settings

Select in the project browser the menu item **Extras--> Options**.
The following window will be opened:



Online settings:

- Query if resource is not up to date.

When trying to download a resource that needs re-compilation, **OpenPCS** will prompt for recompilation.

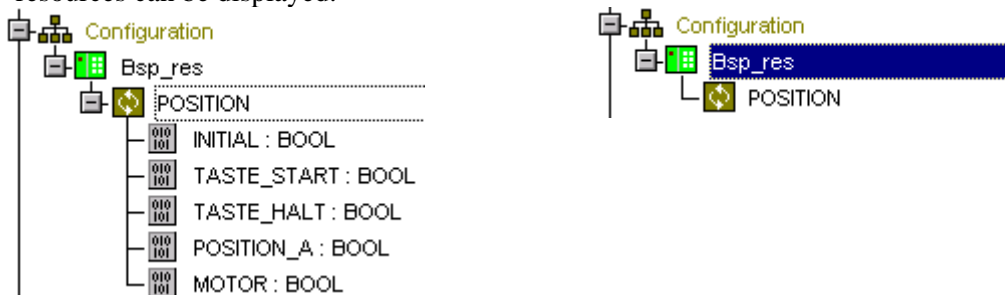
- Query if the program on PLC is not up to date.

If the code on the control system is not the same as the code of the resource which you want to load, you will be asked if you want to transfer the new code.

Browser settings:

- Hide variable nodes:

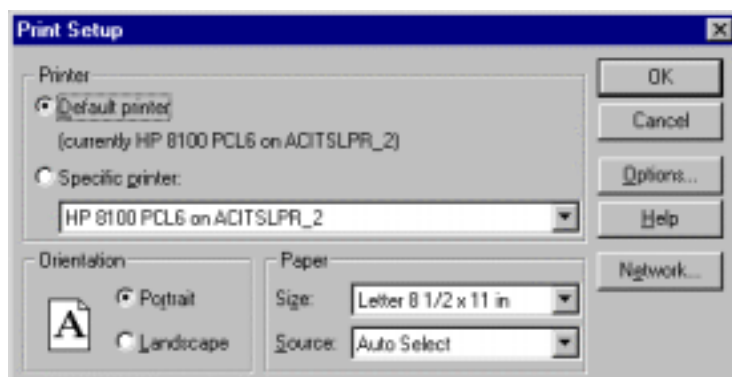
The listing of variables of a resource block can be shown or hidden. Only variables of compiled resources can be displayed.

**Print Setup**

Before you can print, you must set up a printer. That means you must give the information which printer should be used, and how the file should be printed.

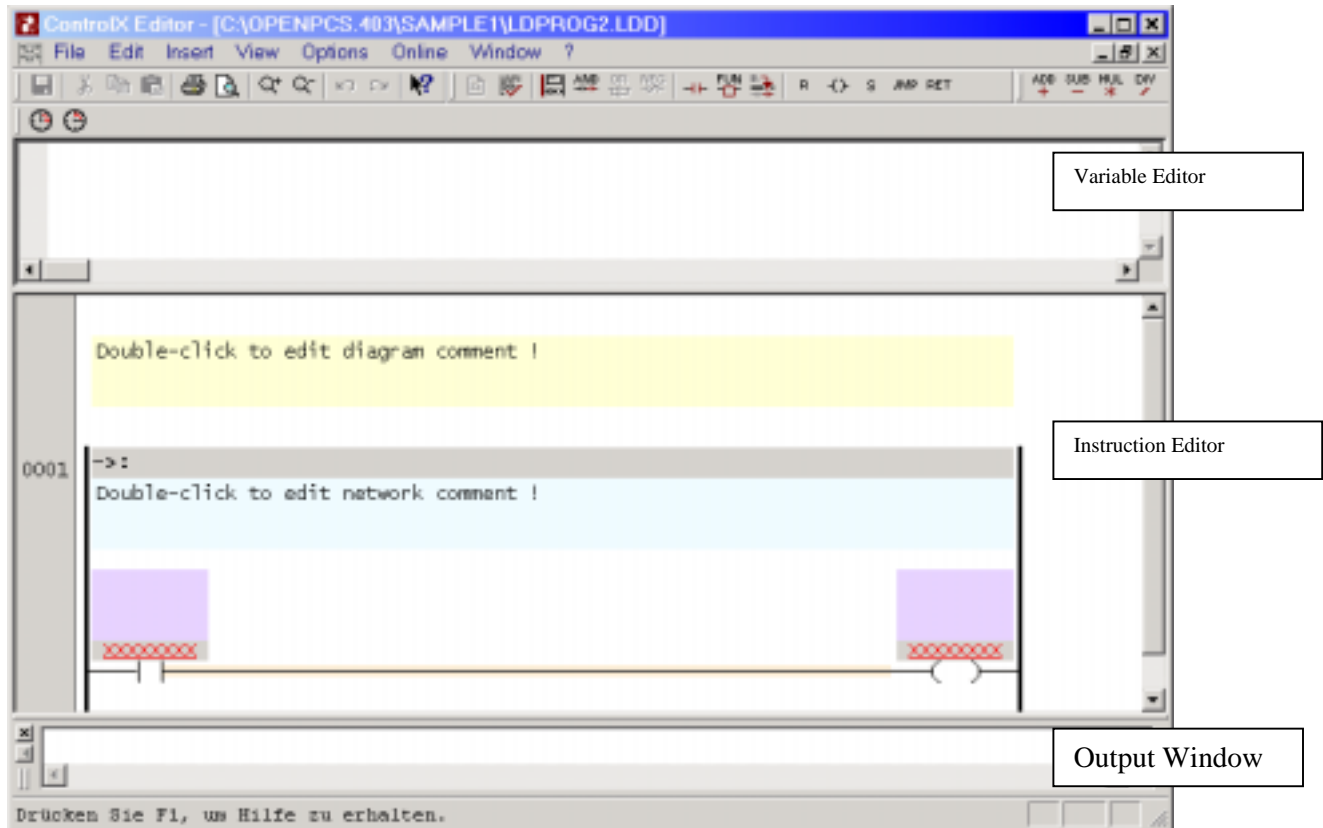
- Select the menu item **File--> Printer Setup ..**

The window **Printer settings** appears (please note that the layout and contents of this box may vary widely depending on the Windows- and network-version you are using):



CONTROL X EDITOR

The Control X Editor is used to write the Ladder program or Function Block or SFC program. The Control X Edit Framework consists of two independent windows. The first window is divided into the **variable editor** in the upper section. All the variables are to be declared in the Variable Editor. The middle section is **instruction editor**. The instruction editor is used to write the program segments. The **output window** is a part of the Control X editor framework. It displays the results of the syntax check or the errors from the compilation. The Syntax Check can be done by Alt+F10 key or by selecting *File → Check Syntax*.



Variable Editor

All the variables used by Ladder or Function Block or SFC program elements must be declared before compilation.

The variables of each type have to be declared within a separate declaration block. It is advisable to separate the different components of a declaration line with tabs. Each declaration block is introduced and closed with a certain key word, e.g. VAR and END_VAR for local variables.

Note that the declaration blocks have to follow in this order.

1. VAR_INPUT
2. VAR_IN_OUT
3. VAR_OUTPUT
4. VAR_GLOBAL
5. VAR_EXTERNAL

6. TYPE

7. VAR

The keyword that introduces and closes the variable blocks is displayed in blue color. A typical declaration will look like...

VAR

(* MyVar PMAC style P variable declaration *)

MyVar AT %IP100 :INT24;

END_VAR

A comment begins with an opening bracket and an asterisk "(" and ends with an asterisk and a closing bracket "*". Comments are displayed in green colour.

Overview of the Declaration Type

Keyword	Usage
VAR	Local Variables, only accessible in the defining program.
VAR_GLOBAL	Global Variable, accessible from all parts of a program. Any other program wishing to access this variable has to declare as external
VAR_EXTERNAL	Declaration for a variable defined as global elsewhere
VAR_INPUT	Input variable, only to be read from within the defining program and to be written by other program. (Typically the calling program).
VAR_OUTPUT	Output variable, to be written by the defining program and read by others. (Typically the calling program)
VAR_IN_OUT	IN_OUT variable; in contrast to input and output variables, IN_OUT variables are passed "by reference", not "by value", i.e. the defining program is handed a reference to the original variable by it's calling program. It may be read and written by the defining program.
TYPE	Definition of local datatype.

Local and global variable declarations can be attributed with the following attributes:

Keyword	Usage
CONSTANT	Use CONSTANT to flag variables that should not be written by the program. Using this keyword makes your program more self-documenting and helps you spot programming errors, if unintentionally you tried to write this variable
AT	Use AT to map variables to physical addresses

Example: Using CONSTANT

Declaration of a remnant variable, a remnant function block instance, and a constant variable:

```
VAR_GLOBAL CONSTANT
    NullKelvin : INT := -273;
END_VAR
```

There are certain restrictions on the variable types you may use in the three different POU types. There is only one variable type that may be used more than once.

Use of the variable types	FU	FB	PRG
Var_Input	▲	▲	
Var_Output		▲	
Var_In_Out		▲	
Var_Global			▲
Var_External		▲	
Var	▲ ▲	▲ ▲	▲ ▲
Type	▲ *	▲ *	▲ *

▲ use only once possible
▲ ▲ use several times possible
* only local inside a POE

Instruction Editor

The instruction section of the Editor is subdivided into so called networks, which help structuring the graphic.

A network consists of:

- Network label
- Network comment
- Network graphic

Network Label

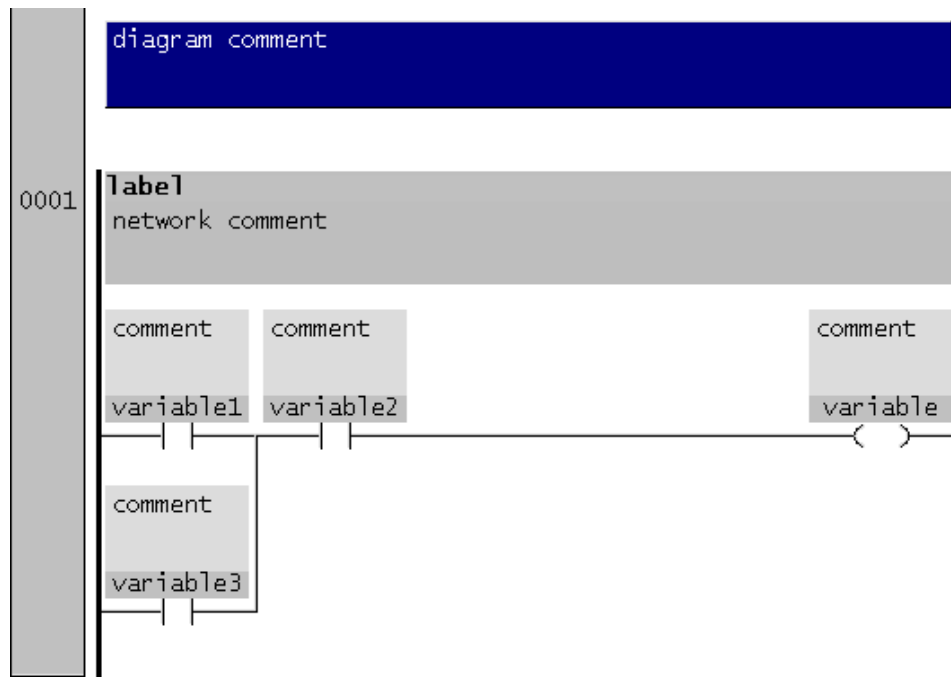
Each network that may be a jump target from within another network will be assigned automatically a preceding alphanumerical identifier or an unsigned decimal integer. By default, networks will be numbered. This numbering of all networks will be updated automatically whenever a new network is inserted. The numbering simplifies finding a certain network corresponds to line numbers of textual programming languages.

Network Comment:

The **Network Comment** is represented as a square area in the ladder diagram. To enter a commentary text, double click on this square. The comment is always placed below the network label. Note that the first network additionally contains a ladder diagram comment above the network label and the network comment.

Network Graphic

The network graphic consists of graphical objects, which may be graphical symbols or connections. Connections transport data between graphical symbols, which process the data at their inputs and transfer the processed data to their outputs. Note that the connections may also cross.



Instructions

Operators

Within a ladder diagram, the term operator designates the graphical objects contact, coil and jump.

Contacts: A contact associates the value of an incoming connection with the value of an assigned variable. The kind of association depends on the type of contact. The result value will be transferred to the connection on the right hand side. There are triggers and interruptors (The boolean value of the variable will not be changed).

Coils: Coils serve to assign values to output variables of networks. The coil saves a function of the state or the transition of the left connector into a boolean variable.

Jump: Jumps manipulate the control flow of programs. They make it possible to directly invoke certain networks in a defined order. When encountering a jump operator, control flow continues at a different network. Thus, jumps are an exception from the basic principle that networks are always processed in a top down fashion.

Functions and Function Blocks

Functions: If a program block might be useable for different and reoccurring parts of a control task, you should consider using a function for this subtask. A function may have more than one input parameter, but one output parameter only. I.e. the calculation

may yield one data element as result only. Functions may not have an internal state, i.e. they may not save any data. Thus, if a function is invoked twice with the same input parameters, it will always yield the same return value.

Function Blocks: You may also use function blocks for reoccurring subtasks. Function blocks may have more than one input parameter and more than one output parameters, too. A function block can save its variable values from one invocation to the next. These values can be used in the next invocation if they are not assigned new values. Thus, function blocks may have an internal state.

Logical Connections

A logical connection is represented as parallel or serial combinations of different networks. There is no restriction on the number of serially or parallelly linked networks. A serial sequence is called an AND connection, while a parallel sequence is called an OR connection.

Adding/Editing Ladder Program

Use the file .LDD created by "Project Browser" (Refer - Creating a program section). Double click the file with the left mouse button in the Project Browser. This opens the file into the Ladder Diagram Editor. If the Control X Editor Framework has not yet been opened, it will be opened together with the file. Use the menu entries and the toolbar to edit the file.

When a new file is being opened, a standard network will be displayed with a contact on the left-hand side and a coil on the right hand side. The label fields of network elements contain "X" by default. The contact and coils are declared as Boolean. Start adding logic.

Inserting New Network

Select **Insert → Network** or (F12 Key) to insert the standard network at current location.

Inserting a Logical AND

Use the Tool bar, select **Insert → AND** or Right Mouse selecting **AND** to insert a relay contact.

Inserting a Logical OR

First hold the SHIFT key down selecting the OR rung and then use the Tool bar or select **Insert → OR** or Right Mouse selecting **OR** to insert an OR relay contact.

Inserting a Parallel Coil or JMP

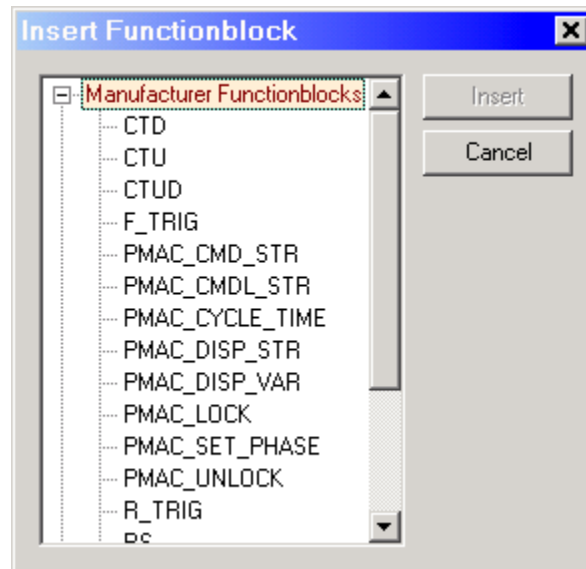
First select the rung just before the current Coil or JMP. Then use the Tool bar or select **Insert → Coil** or Right Mouse, selecting **Coil** to insert a parallel Coil.

Inserting a Function Block

Select the desired rung area and then **Insert → Functionblock** to insert the function block at current location.

The list will be available. It consists of Manufacturer function block and user function block. The explanation of the Manufacturer function block is in the PMAC LADDER Reference Section. Select the required function block and press **Insert** to close the box.

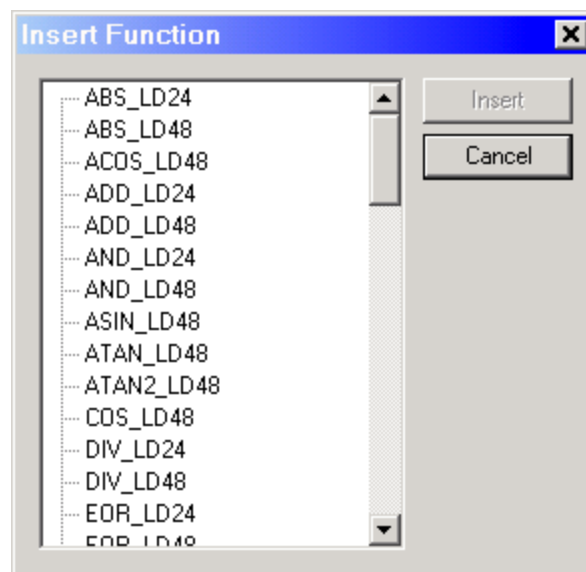
The User functionblocks are the function blocks created by user.



Inserting a Function

Select **Insert** → **Function** to insert the function at the currently selected location.

The list will be available. It consists of all the available PMAC functions. Select the required function and press **Insert** to close the box. A detailed explanation of these functions is available in PMAC Ladder Reference Section.



Inserting a Variable

After declaring the variable in the Variable Edit section, select the undefined XXXXX variable with a right mouse click. Then select **Insert Variable**. A list of variables will be displayed for inserting. You can also edit the XXXXX directly.

Saving and checking the Syntax of Your Program

Save the program by pressing...

CNTL+S or **File → Save**.

After saving the file, check the syntax. The Syntax Check can be done by **Alt+F10** key or by selecting **File → Check Syntax**.

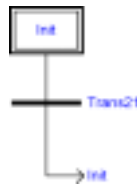
Double clicking the error will identify where the error is in the ControlX editor.

Adding/Editing SFC Program

Use the file .sfc created by “Project Browser” (Refer - Creating a program section). Double click the file with the left mouse button in the Project Browser. This opens the file into the SFC Editor. If the Control X Editor Framework has not yet been opened, it will be opened together with the file.

Use the menu entries and the toolbar to edit the file.

Every new document consists of a minimal SFC-program. That means it consists of an initial step, a transition, and a finishing jump back to the initial step.



Steps and Initial Steps

The code of a step is executed cyclic if and only if this is an active state. In principle, one can say that the code is surrounded by a loop, which is entered if a previous transition switches, and is left if a following transition switches. When a step is activated, its code is executed at least one time. Initial steps are always active at program start that means that no preceding transition is necessary. In standard, the entry-point into the IL-program is the first IL-element. Every step can be converted into an initial step by activating the control box “initial step” in the properties window. De-activating this switch will turn the initial step back into a normal step.

The name of a step must meet following syntax:

The first character of the step name is a letter (,a‘-,z‘, ,A‘-,Z‘); every further character is a letter or a number (‘0‘ – ‘9‘) or a underline (,_‘).

Valid step names: „Step1“ „S_1“ „S1_“

Invalid step names: „_Step1“ „1Step“ „Heater off“

Step names have a maximum length of 31 characters.

To add STEP and Transition select the element where it needs to be inserted and select

Insert **→Step/Transition** to insert at current location.

To add STEP and Transition to the left side of the element, select the element where it needs to be inserted (using the **SHIFT and CNTRL** keys) and select Insert **→Step/Transition Left** to insert to the left of the current location.

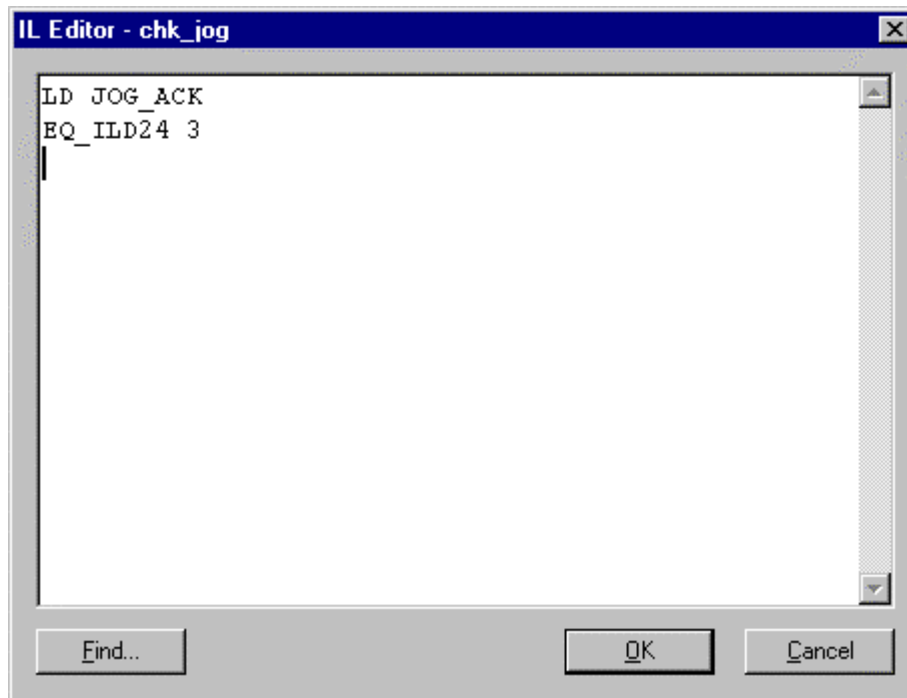
To add STEP and Transition to the right side of the element, select the element where it needs to be inserted (using the **SHIFT and CNTRL** keys) and select Insert **→Step/Transition Right** to insert to the right of the current location.

Transitions

Transitions are responsible for the change of the active state of previous step(s) to the following step(s).

Transitions show the possible change in form of a true, Boolean statement (transition condition).

The code of the transition has to be written so that the current result at the end of the code is of type BOOL. The transition switches if and only if the value of the accumulator is TRUE.



Jumps

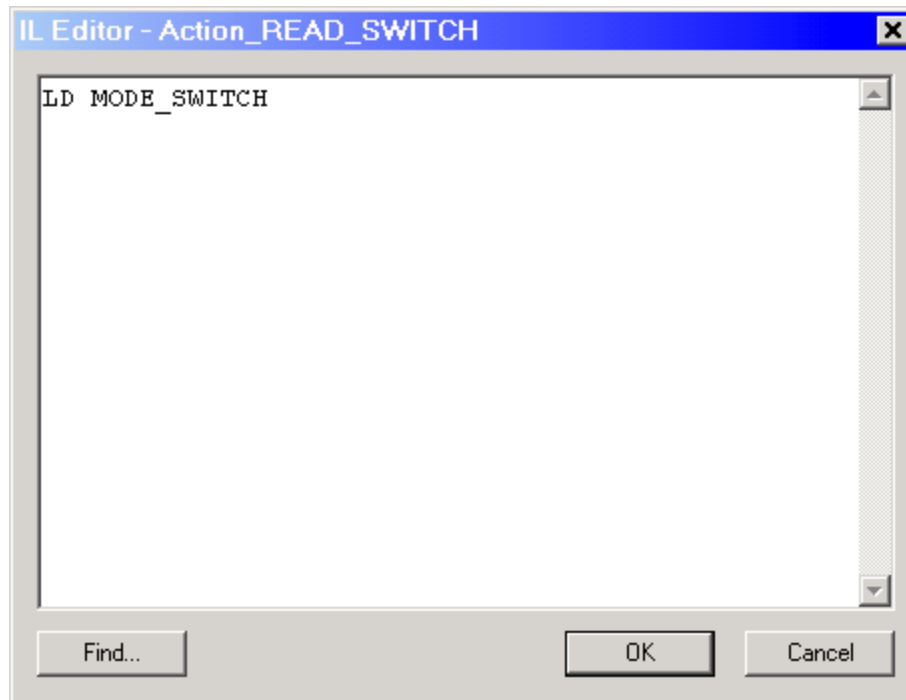
Jumps are elements of a SFC-plan for controlling the flow of execution. With the up to now introduced elements, the activation of the steps happens always from top to bottom. For programming of cycles and similar things, a further possibility is necessary to activate previous steps. Jumps exist to provide this functionality.

The predecessor of a jump element is always a transition. The target of a jump is always a step. The target of the jump is fixed by giving the jump the same name as the selected target-step. If a step is given as a target of a jump, its name must be unique. If a jump-target is not or more than once available, corresponding error messages are created during the syntax control.

To guarantee the consistency of SFC-plan, the insertion of a jump is possible only as the last element of a divergence of sequence selection.

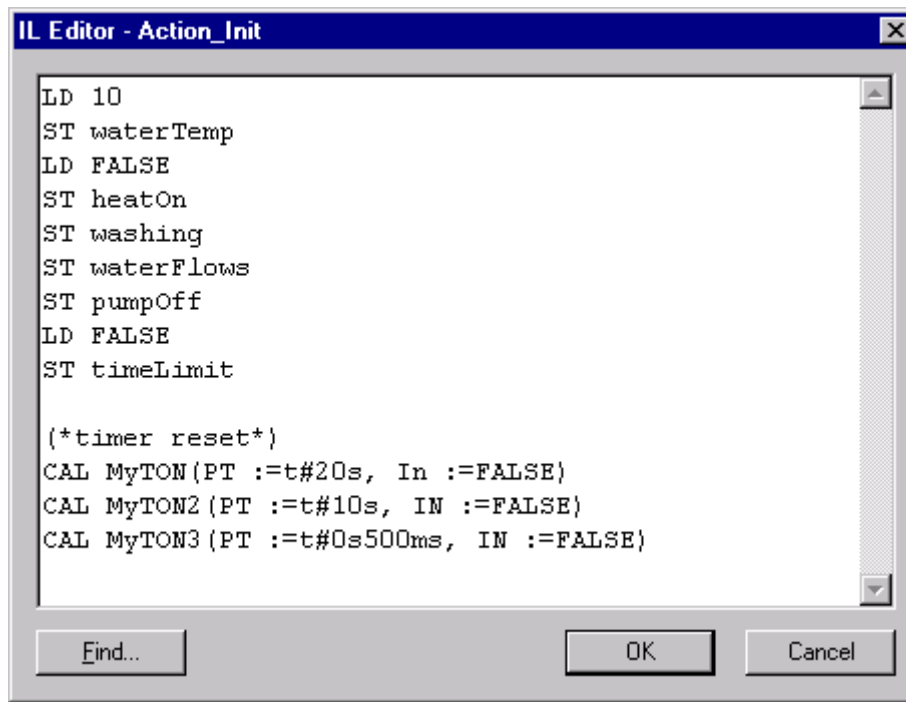
Editing of the (Step/Transition) Program Blocks

Steps and transitions need the IL-code fragments for their functionality. To edit the code, a text editor can be opened by **Edit → Edit** when the corresponding element was marked or double click the element. In this editor version, the code of transitions and steps can be given as IL-code only. A list of available of IL code Functions are in PMAC LADDER Reference Section: PMAC LADDER SFC INSTRUCTION LIST (IL) FUNCTIONS. A list of the available Function Blocks are in PMAC LADDER LD & SFC FUNCTION BLOCKS.



The names and the comment lines of an element can be adjusted by the appearing dialog under **Edit → Properties** or by clicking right mouse button on the element. The context menu of SFC-element can call both named functions equivalently. The context menu can be opened by a click with the right mouse button on the appropriate element.

To call function block from SFC program use **CAL <FBD Instance>** command from IL editor.



The Structure of an IL-Line

An IL-line has the following form, when optional parts are set in [square] brackets, and expressions are set between <sharp> brackets:

```
[<Label>:] <Operator> <Operand1> [,<Operand2>,<Operand3>,...] [(  
<Comment> *)]
```

At the beginning is a label if the line represents a jump target. After that an operator is placed, followed by the operands and separated by commas. Comments are enclosed by (* and *).

Example:

VAR

```
a : INT24;  
b : INT24;  
c : INT24;  
d : INT48;  
e : INT48;  
f : INT48;
```

END_VAR

```
Start: LD a      (* Load a in the register *)  
ADD_IL24 b      (*PMAC 24 bit integer Add b to the register *)  
ST c            (* store result to variable c *)  
LD d            (* Load d in the register *)  
MUL_IL48 e      (*PMAC 48 bit float MUL e to the register *)  
ST f            (* store result to variable f *)
```

A call to a function block instance is done using operator CAL and CALC respectively; the oper and is the instance name, followed by arguments supplied in parentheses:

```
[<Label>:] CAL/CALC <Instance name>(  
    [<Input1>:=<Value1>,<Input2>:=<Value2>,...]  
    |  
    [<Variable1>:=<Output1>,<Variable2>:=<Output2>,.  
    ..]  
)
```

The parameter transfer consists of two parts. In the first part the parameters are transferred to the function block by setting values to the INPUT- and IN_OUT-variables respectively. The variables, which get no value, retain the value of their last call and their initial value respectively. Separated by a '|' from the first part, output parameters are specified.

Example:

In the declaration part:

VAR

```
FB_TON: TON; (* Declaration of the function block instance FB of type TON *)  
a : BOOL;  
b : TIME;  
c : BOOL;  
d : TIME;
```

END_VAR

In the instruction part:

```
CAL FB_TON (  
    IN :=a,  
    PT :=b  
    |  
    c:=Q,  
    d:=ET  
)
```

(* Call of the function block instance FB with input-parameter a and b, and output-value-transmission to c and d *)

```
CAL Counter_3(  
    CU :=Photosensor  
assembly,  
    RESET:=RES_Button,  
    PV := 120  
    |  
    full := Q  
)
```

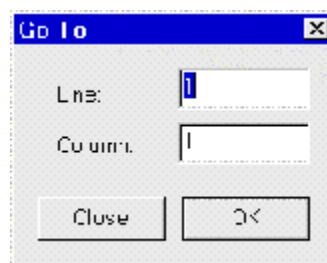
Syntax control

The syntax control, integrated in the SFC-editor, tests the entire plan for logical consistency. Therefore, for example, jump-targets are tested for uniqueness or names of plan elements for validity.

The syntax check is invoked during **SAVING** of the chart. If errors are detected during the control, no IL-code is created! To save press **CNTL+S** or **File → Save**.

By double click on the error lines, which appear in the output window, elements of the plan that cause this error or are in connection can be marked.

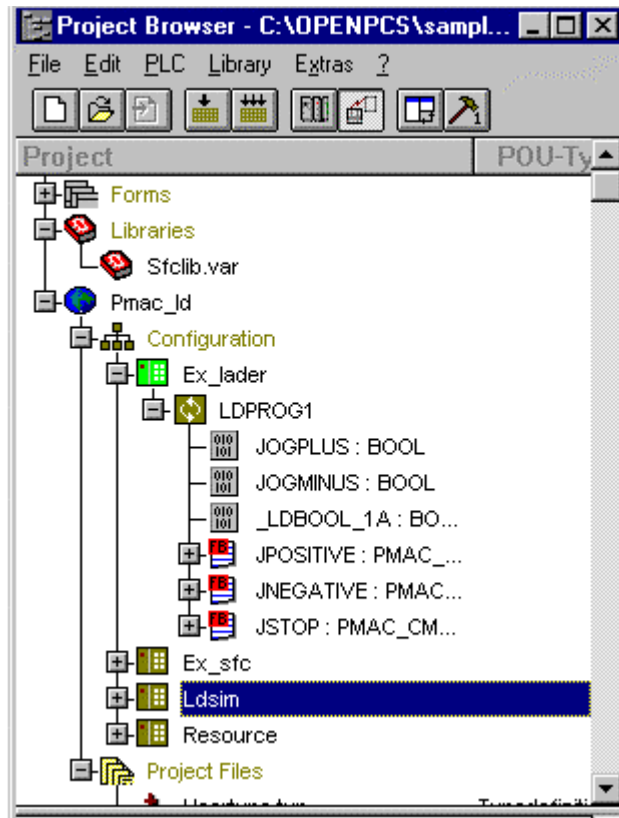
No IL-syntax errors are recognised during the syntax check. If mistakes are made in the variable declaration, in the code of the single steps or in the formulation of the transition conditions, these could be recognised only during the compilation in OpenPCS. Then, the compiler lists the corresponding line numbers of the POU-file which caused an error. In order to find a line number of the POU-file to an element of the SFC-plan, Edit--> Goto IL-Line can be used to find this location in the SFC chart



PMAC LADDER – EXAMPLE

Open Project

Open the Samples **Ladder/SFC Sample** at the start of the program.



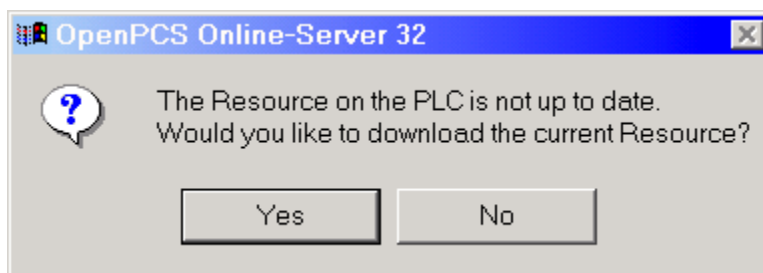
The Project name is **Pmac_Id**. The files under **Configuration** are the resource (SmartPLC or PMAC) files. In this example it is SmartPLC. If **Ex_Ladder** is not highlighted, right mouse click selecting **Set active resource**. The declarations under LDProg1 are the variables and function block used in the program.

Compile / Build Project

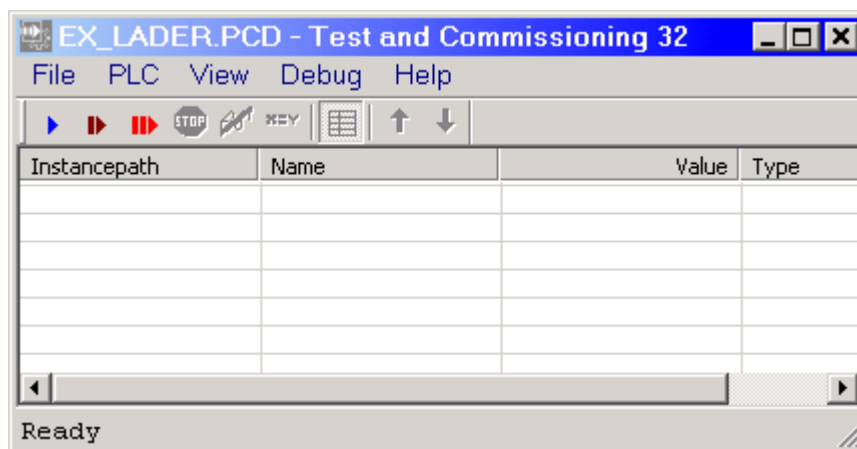
To compile/Build the project keep **Ex_Ladder** selected and select **PLC → Build or Build All** from browser menu. **Build all** will compile all the project files whereas **build** will compile the change file. For the example select **Build All**. This will compile the project and result will be displayed in Output window.

Go Online

This step is necessary to run PLC in PC simulator SmartPLC. Select **PLC → Online**. This will start the download process. Select **YES** to download the PLC.







The Test and Commission window will be open.



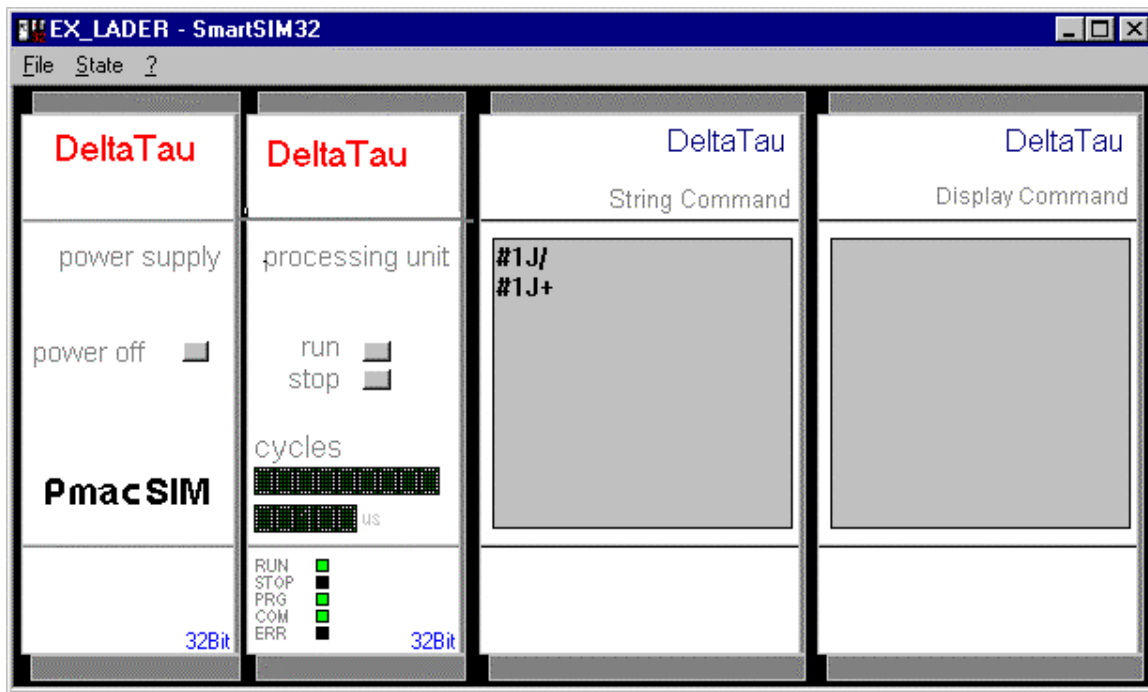
Starting and Stopping the Program

Press the Coldstart button from Test and Commission 32 window.

	<i>PLC → STOP</i>	Select <i>PLC → Stop</i> to immediately stop the program
	<i>PLC → Coldstart</i>	Select <i>PLC → Coldstart</i> to perform a cold start. All variables will be initialized to the initial value as programmed.
	<i>PLC → Warmstart</i>	Select <i>PLC → Warmstart</i> to initialize all normal variables, but keep the values of all variables programmed as RETAIN.
	<i>PLC → Hotstart</i>	Select <i>PLC → Hotstart</i> to continue execution where it stopped, not initializing any variable.

Watching Variables

To watch JOGPLUS variable in the watch window, double click on this variable in the Project Browser window. The JOGPLUS will be displayed in the Test and commission 32 window. If the PLC is running then its status will be displayed immediately. In the Test and Commissioning window, double click JOGPLUS and set it to "1." You should see the following in the SmartSIM32 window.

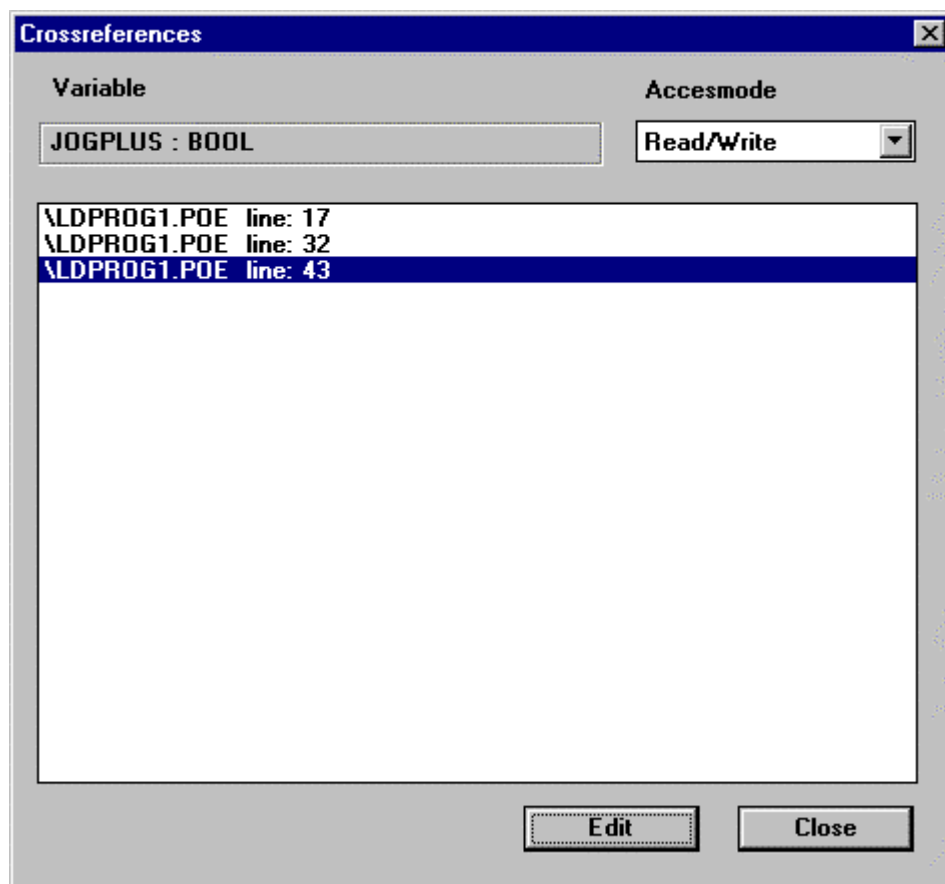


Watching Power Flow

To watch power flow, select **LDPROG1**. Click right mouse button and select **Open**. This will start the control X Editor and will load the selected file. (**LDPROG1**) To start monitoring the power flow select **OnLine** → **StatusDisplay** in the editor.

CrossReference

In the Project Browser right mouse click the JOGPLUS variable and select **Crossreference**. This produces the list of the variable used in the project. This feature is used to go quickly to the correct spot of the power flow for the selected variable. Clicking the mouse will put you in the Power Flow window where the JOGPLUS variable is being used.



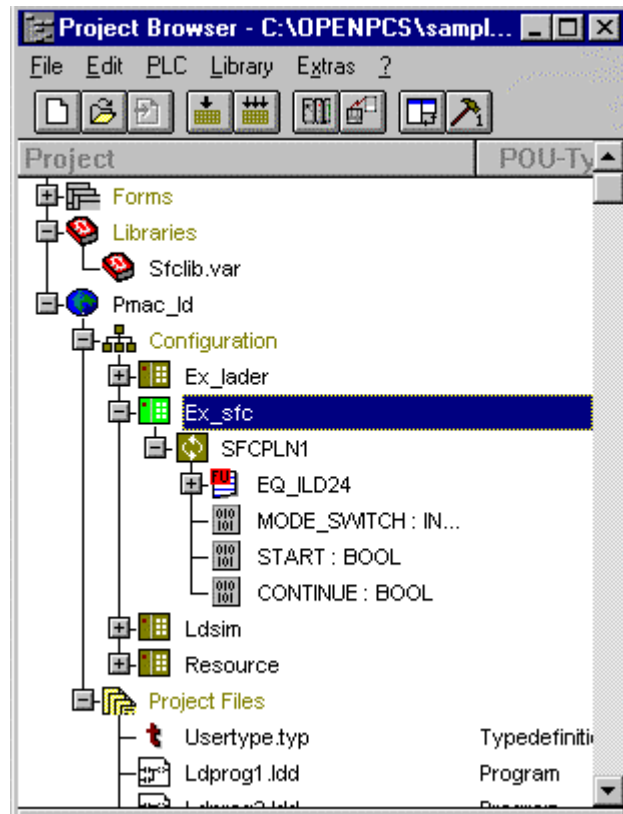
Example Output

Changing the states of JOGPLUS and JOGMINUS will show the command that will be sent to the PMAC in the **SmartSIM32 Delta Tau String Command** window.

SFC PROGRAM - EXAMPLE

Set the Active Resource

Highlight **Ex_SFC** and right mouse click selecting **Set active resource**. The declarations under SFCPLN1 are the variables and function block used in the program.



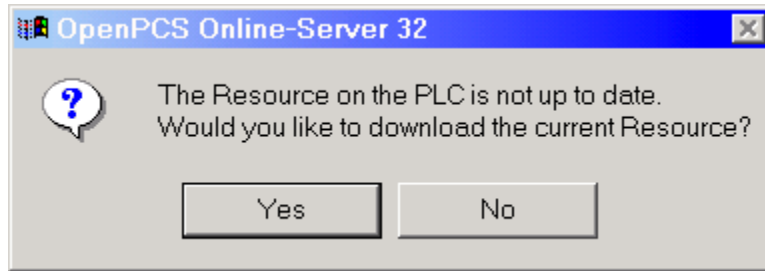
Compile / Build Project

Select **PLC → Build or Build All** from browser menu. Build all will compile all the project files whereas **build** will compile the changed files. For the example select **Build All**. This will compile the project and result will be displayed in Output window.

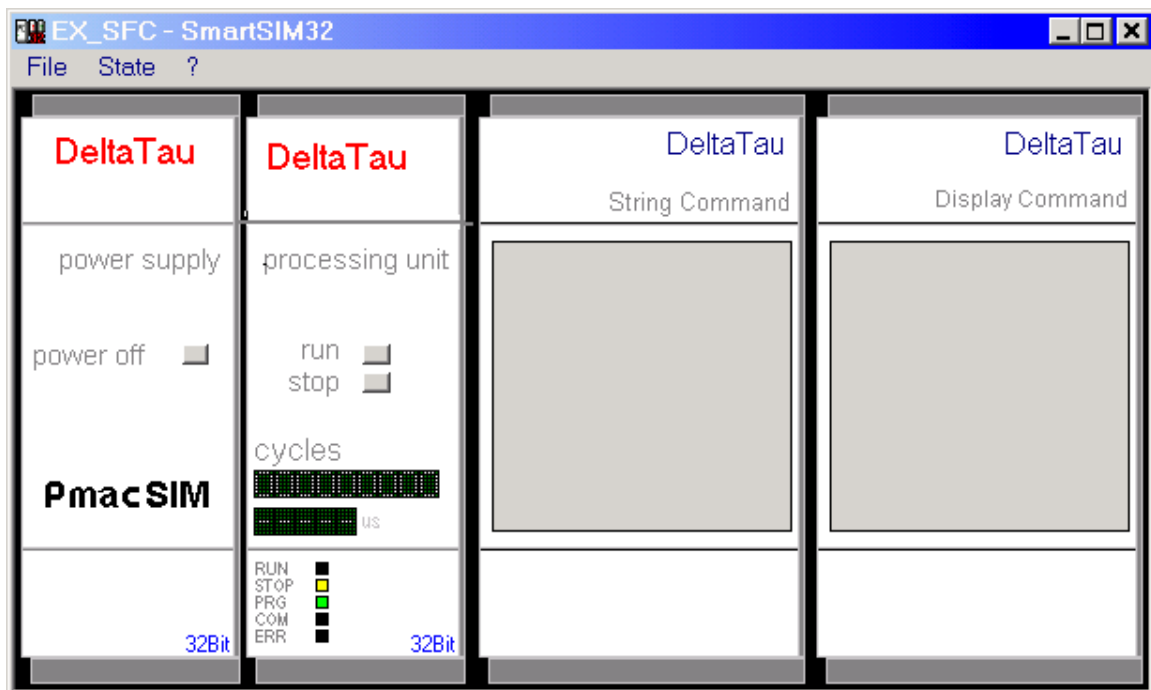
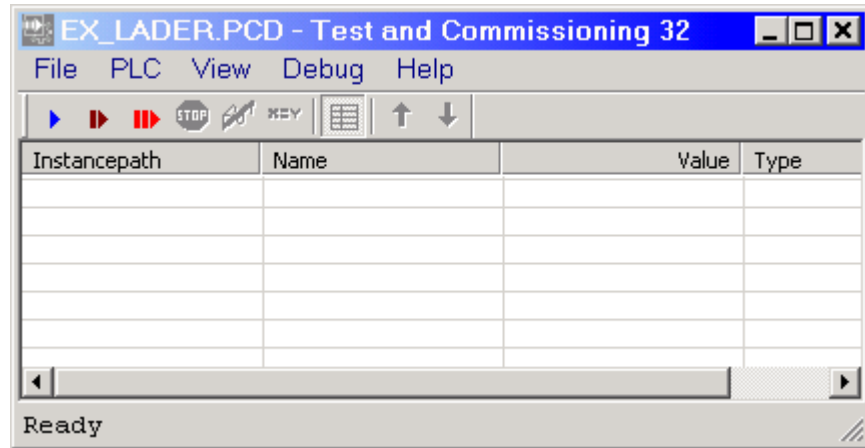
Go Online

Select **PLC → Online**.

As this is in simulation mode, select **YES** to download the PLC to PC simulator.







The Test and Commission widow will be open along with the simulation window.



Starting and Stopping the Program

Press the Coldstart button from Test and Commission 32 window.

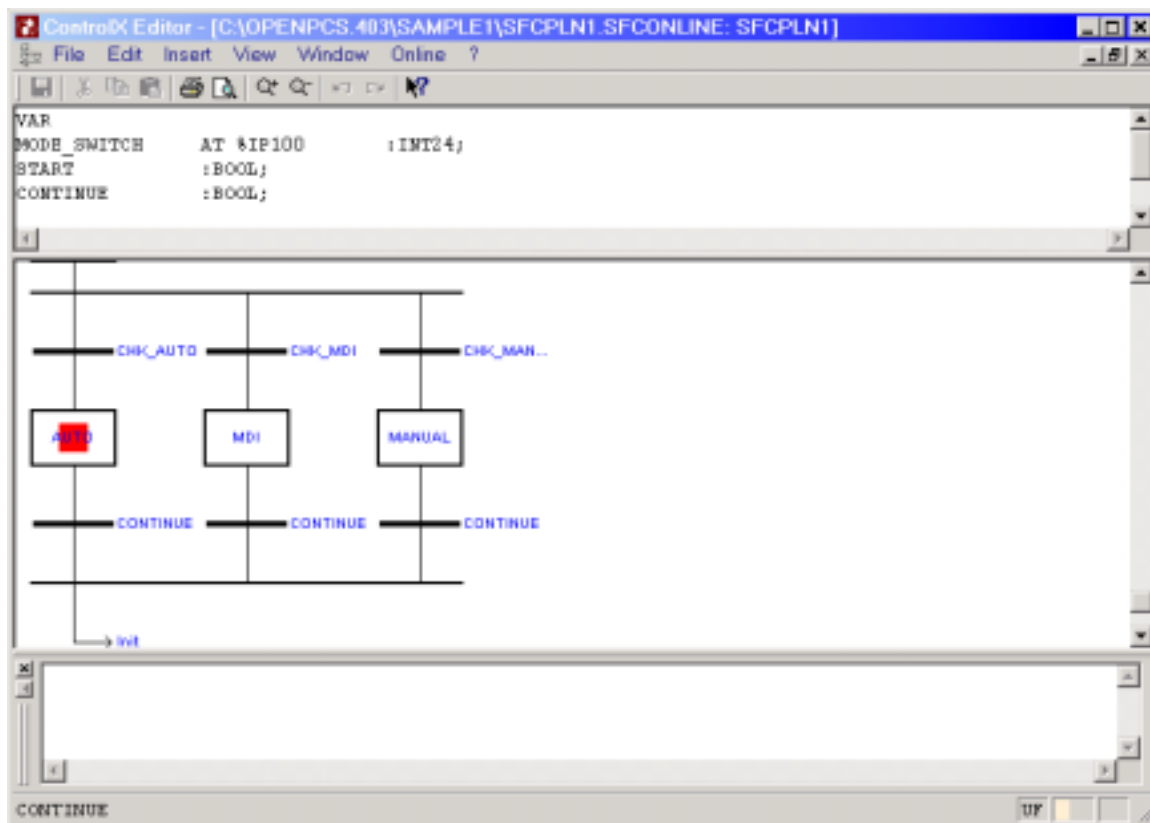
	PLC →STOP	Select PLC →Stop to immediately stop the program
	PLC →Coldstart	Select PLC →Coldstart to perform a cold start. All variables will be initialized to the initial value as programmed.
	PLC →Warmstart	Select PLC →Warmstart to initialize all normal variables, but keep the values of all variables programmed as RETAIN.
	PLC →Hotstart	Select PLC →Hotstart to continue execution where it stopped, not initializing any variable.

Watching Variables

To watch or set variable in watch window, double click on MODE_SWITCH variable. The MODE_SWITCH will be displayed in the Test and commission 32 window. If the PLC is running then its status will be displayed immediately. To set the value double clicks on this variable and set the value. For our example it takes 1,2 or 3 as input value.

Watching Power Flow

To watch power flow, select **SFCPLAN1**. Click right mouse button and select **Open**. This will start the control X Editor and will load the selected file. (**SFCPLAN1**) To start monitoring the power flow select **OnLine →Display** in the editor. When the value of MODE_SWITCH is set to 1 the power flow will display AUTO mode. Value 2 is for MDI mode and 3 is for MANUAL mode.



PMAC LADDER REFERENCE SECTION

In order to provide maximum computational throughput, PMAC Ladder chose to use non-standard IEC-61131 data types for almost all functions. These data types fit naturally with the data type used by the PMAC DSP processor chip and its floating point library. The non-standard data types are INT24 (24 bit integer) and REAL48 (PMAC's 48 bit floating point). This provided us with an estimated 2 to 3 times throughput improvement over using the standard IEC 16/32 bit integer and 32 bit floating point. The following table provides a list of the available functions when developing Ladder Programs. From these basic building blocks, you should be able to develop your own User Defined Function Blocks.

PMAC Ladder LD Functions

PMAC Ladder LD Functions	Data Type	Description
Arithmetic		Standard Arithmetic Functions
ABS_LD24	INT24	PMAC 24 bit Integer ABS (Absolute Value)
ADD_LD24	INT24	PMAC 24 bit Integer ADD
SUB_LD24	INT24	PMAC 24 bit Integer SUB
MUL_LD24	INT24	PMAC 24 bit Integer MUL
DIV_LD24	INT24	PMAC 24 bit Integer DIV
MOD_LD24	INT24	PMAC 24 bit Integer MOD (Modulo)
NEG_LD24	INT24	PMAC 24 bit Integer NEG (Negate)
ABS_LD48	REAL48	PMAC 48 bit Floating Point ABS
ADD_LD48	REAL48	PMAC 48 bit Floating Point ADD
SUB_LD48	REAL48	PMAC 48 bit Floating Point SUB
MUL_LD48	REAL48	PMAC 48 bit Floating Point MUL
DIV_LD48	REAL48	PMAC 48 bit Floating Point DIV
MOD_LD48	REAL48	PMAC 48 bit Floating Point MOD
NEG_LD48	REAL48	PMAC 48 bit Floating Point NEG
Logical		Standard Logical Functions
AND_LD24	INT24	PMAC 24 bit Integer AND
OR_LD24	INT24	PMAC 24 bit Integer OR
EOR_LD24	INT24	PMAC 24 bit Integer EOR (Exclusive OR)
NOT_LD24	INT24	PMAC 24 bit Integer NOT (1'S Compliment)
ROL_LD24	INT24	PMAC 24 bit Unsigned Integer Logical Rotate Left
ROR_LD24	INT24	PMAC 24 bit Unsigned Integer Logical Rotate Right
SHL_LD24	INT24	PMAC 24 bit Unsigned Integer Logical Shift Left
SHR_LD24	INT24	PMAC 24 bit Unsigned Integer Logical Shift Right
AND_LD48	REAL48	PMAC 48 bit Floating Point Logical AND
OR_LD48	REAL48	PMAC 48 bit Floating Point Logical OR
EOR_LD48	REAL48	PMAC 48 bit Floating Point Logical EOR
Arithmetic		Complex Arithmetic Functions
SIN_LD48	REAL48	PMAC 48 bit Floating Point SIN (SIN value in Degrees)
COS_LD48	REAL48	PMAC 48 bit Floating Point COS (COS value in Degrees)
TAN_LD48	REAL48	PMAC 48 bit Floating Point TAN (TAN value in Degrees)
ASIN_LD48	REAL48	PMAC 48 bit Floating Point ASIN (ASIN return in Degrees)
ACOS_LD48	REAL48	PMAC 48 bit Floating Point ACOS (ACOS return in Degrees)
ATAN_LD48	REAL48	PMAC 48 bit Floating Point ATAN (ATAN return in Degrees)
ATAN2_LD48	REAL48	PMAC 48 bit Floating Point ATAN2 (2 Input ATAN return in

		Degrees)
SQRT_LD48	REAL48	PMAC 48 bit Floating Point SQRT (Square Root)
EXP_LD48	REAL48	PMAC 48 bit Floating Point EXP (e to the X)
LN_LD48	REAL48	PMAC 48 bit Floating Point LN (Natural LOG base e)
LOG_LD48	REAL48	PMAC 48 bit Floating Point LOG (LOG base 10)
TRUNC_LD48	REAL48	PMAC 48 bit Floating Point Truncate down to the nearest Integer Value
Limit Functions		Limit Functions
MAX_LD24	INT24	PMAC 24 bit Integer MAX (Maximum Value)
MIN_LD24	INT24	PMAC 24 bit Integer MIN (Minimum Value)
MAX_LD48	REAL48	PMAC 48 bit Floating Point MAX (Maximum Value)
MIN_LD48	REAL48	PMAC 48 bit Floating Point MIN (Minimum Value)
Assignment		Assignment Functions
MOVE_LD24	INT24	PMAC 24 bit Integer MOVE (Assignment)
MOVE_LD48	REAL48	PMAC 48 bit Floating Point MOVE (Assignment)
Comparison		Arithmetic Comparison Functions
EQ_ILD24	INT24	PMAC 24 bit Integer EQ (Equal Compare)
GE_ILD24	INT24	PMAC 24 bit Integer GE (Greater Than or Equal Compare)
GT_ILD24	INT24	PMAC 24 bit Integer GT (Greater Than Compare)
LE_ILD24	INT24	PMAC 24 bit Integer LE (Less Than or Equal Compare)
LT_ILD24	INT24	PMAC 24 bit Integer LT (Less Than Compare)
NE_ILD24	INT24	PMAC 24 bit Integer NE (Not Equal Compare)
EQ_ILD48	REAL48	PMAC 48 bit Floating Point EQ (Equal Compare)
GE_ILD48	REAL48	PMAC 48 bit Floating Point GE (Greater Than or Equal Compare)
GT_ILD48	REAL48	PMAC 48 bit Floating Point GT (Greater Than Compare)
LE_ILD48	REAL48	PMAC 48 bit Floating Point LE (Less Than or Equal Compare)
LT_ILD48	REAL48	PMAC 48 bit Floating Point LT (Less Than Compare)
NE_ILD48	REAL48	PMAC 48 bit Floating Point NE (Not Equal Compare)
Conversions		Data Type Conversion Functions
LD24_TO_LD48	INT24/REAL48	PMAC 24 bit Integer TO 48 bit Floating Point
LD48_TO_LD24	REAL48/INT24	PMAC 48 bit Floating Point TO 24 bit Integer
LD24_TO_INT	INT24/INT	PMAC 24 bit Integer TO 16 bit Integer
INT_TO_LD24	INT/INT24	PMAC 16 bit Integer TO 24 bit Integer
GREY4_TO_LD24	GREY4/ INT24	PMAC 24 bit Integer 4 bit Grey Code TO 24 bit Integer
Time		Time Functions
GetTime_LD48	REAL48	PMAC 48 bit Floating Point Get Time (milli-seconds) since Power On

Standard Arithmetic Functions

ABS_LD24

VAR_INPUT

EN : BOOL;

IN : INT24;

VAR_OUTPUT

ENO : BOOL;

OUT : INT24;

PROCESS: (Done when EN = TRUE)

ENO = EN;
OUT = ABS(IN);

//**

ADD_LD24

VAR_INPUT

EN : BOOL;
IN1 : INT24;
IN2 : INT24;

VAR_OUTPUT

ENO : BOOL;
OUT : INT24;

PROCESS: (Done when EN = TRUE)

ENO = EN;
TMP = IN1 + IN2;
If ((TMP < 8388607) && (TMP >= -8388608)) // Test for Operation overflow for 24 bits
OUT = TMP;
Else
ENO = FALSE;

//**

SUB_LD24

VAR_INPUT

EN : BOOL;
IN1 : INT24;
IN2 : INT24;

VAR_OUTPUT

ENO : BOOL;
OUT : INT24;

PROCESS: (Done when EN = TRUE)

ENO = EN;
TMP = IN1 - IN2;
If ((TMP < 8388607) && (TMP >= -8388608)) // Test for Operation overflow for 24 bits
OUT = TMP;
Else
ENO = FALSE;

//**

MUL_LD24

VAR_INPUT

EN : BOOL;
IN1 : INT24;
IN2 : INT24;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **INT24**;

PROCESS: (Done when EN = TRUE)

ENO = EN;
TMP = IN1 * IN2;
If ((TMP < 8388607) && (TMP >= -8388608)) // Test for Operation overflow for 24 bits
OUT = TMP;
Else
ENO = FALSE;

//**

DIV_LD24VAR_INPUT

EN : **BOOL**;
IN1 : **INT24**;
IN2 : **INT24**;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **INT24**;

PROCESS: (Done when EN = TRUE)

ENO = EN;
If (IN2 != 0) // Test for Operation overflow
OUT = IN1 / IN2;
If (OUT == 8388608)
OUT = 8388607 // Max 24 bit positive value
Else
ENO = FALSE;

//**

MOD_LD24VAR_INPUT

EN : **BOOL**;
IN1 : **INT24**;
IN2 : **INT24**;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **INT24**;

PROCESS: (Done when EN = TRUE)

ENO = EN;
If (IN2 != 0) // Test for Operation overflow
OUT = IN1 % IN2;
Else
ENO = FALSE;

//**

NEG_LD24VAR_INPUT

EN : **BOOL**;
IN : **INT24**;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **INT24**;

PROCESS: (Done when **EN** = **TRUE**)

ENO = **EN**;
OUT = **-IN**;

//**

ABS_LD48VAR_INPUT

EN : **BOOL**;
IN : **REAL48**;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **REAL48**;

PROCESS: (Done when **EN** = **TRUE**)

ENO = **EN**;
OUT = **ABS(IN)**;

//**

ADD_LD48VAR_INPUT

EN : **BOOL**;
IN1 : **REAL48**;
IN2 : **REAL48**;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **REAL48**;

PROCESS: (Done when **EN** = **TRUE**)

ENO = **EN**;
OUT = **IN1 + IN2**;

//**

SUB_LD48VAR_INPUT

EN : **BOOL**;
IN1 : **REAL48**;

IN2 : REAL48;

VAR_OUTPUT

ENO : BOOL;

OUT : REAL48;

PROCESS: (Done when EN = TRUE)

ENO = EN;

OUT = IN1 - IN2;

//**

MUL_LD48

VAR_INPUT

EN : BOOL;

IN1 : REAL48;

IN2 : REAL48;

VAR_OUTPUT

ENO : BOOL;

OUT : REAL48;

PROCESS: (Done when EN = TRUE)

ENO = EN;

OUT = IN1 * IN2;

//**

DIV_LD48

VAR_INPUT

EN : BOOL;

IN1 : REAL48;

IN2 : REAL48;

VAR_OUTPUT

ENO : BOOL;

OUT : REAL48;

PROCESS: (Done when EN = TRUE)

ENO = EN;

If (IN2 != 0) // Test for Operation overflow

OUT = IN1 / IN2;

Else

ENO = FALSE;

//**

MOD_LD48

VAR_INPUT

EN : BOOL;

IN1 : REAL48;

IN2 : REAL48;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **REAL48**;

PROCESS: (Done when **EN** = **TRUE**)

ENO = **EN**;
If (**IN2** != 0) **// Test for Operation overflow**
 OUT = **IN1** % **IN2**;
Else
 ENO = **FALSE**;

//*****

NEG_LD48VAR_INPUT

EN : **BOOL**;
IN : **REAL48**;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **REAL48**;

PROCESS: (Done when **EN** = **TRUE**)

ENO = **EN**;
OUT = - **IN**;

//*****

Standard Logical Functions

AND_LD24

VAR_INPUT

EN : **BOOL**;
IN1 : **INT24**;
IN2 : **INT24**;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **INT24**;

PROCESS: (Done when **EN** = **TRUE**)

ENO = **EN**;
OUT = **IN1** & **IN2**;

//*****

OR_LD24

VAR_INPUT

EN : **BOOL**;
IN1 : **INT24**;
IN2 : **INT24**;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **INT24**;

PROCESS: (Done when **EN** = **TRUE**)

ENO = **EN**;
OUT = **IN1** | **IN2**;

//*****

EOR_LD24

VAR_INPUT

EN : **BOOL**;
IN1 : **INT24**;
IN2 : **INT24**;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **INT24**;

PROCESS: (Done when **EN** = **TRUE**)

ENO = **EN**;
OUT = **IN1** ^ **IN2**;

//*****

NOT_LD24VAR_INPUT

EN : **BOOL**;
IN : **INT24**;

VAR_OUTPUT

ENO : **BOOL**;
OUT : **INT24**;

PROCESS: (Done when EN = TRUE)

ENO = **EN**;
OUT = **~IN**; // The One's compliment (0's converted to 1' and 1's converted to 0's)

//**

ROL_LD24VAR_INPUT

EN : **BOOL**;
IN : **INT24**; // Input
N : **INT24**; // Rotate amount

VAR_OUTPUT

ENO : **BOOL**;
OUT : **INT24**;

PROCESS: (Done when EN = TRUE)

ENO = **EN**;
OUT = **ROL(IN)**; // 24 bit word Rotate Left

//**

ROR_LD24VAR_INPUT

EN : **BOOL**;
IN : **INT24**; // Input
N : **INT24**; // Rotate amount

VAR_OUTPUT

ENO : **BOOL**;
OUT : **INT24**;

PROCESS: (Done when EN = TRUE)

ENO = **EN**;
OUT = **ROR(IN)**; // 24 bit word Rotate Right

//**

SHL_LD24VAR_INPUT

EN : **BOOL**;
IN : **INT24**; // Input
N : **INT24**; // Rotate amount

VAR_OUTPUT

ENO : BOOL;
OUT : INT24;

PROCESS: (Done when EN = TRUE)

ENO = EN;
OUT = SHL(IN); // 24 bit word Logical Shift Left

//**

SHR_LD24VAR_INPUT

EN : BOOL;
IN : INT24; // Input
N : INT24; // Rotate amount

VAR_OUTPUT

ENO : BOOL;
OUT : INT24;

PROCESS: (Done when EN = TRUE)

ENO = EN;
OUT = ROR(IN); // 24 bit word Logical Shift Right

//**

AND_LD48VAR_INPUT

EN : BOOL;
IN1 : REAL48;
IN2 : REAL48;

VAR_OUTPUT

ENO : BOOL;
OUT : REAL48;

PROCESS: (Done when EN = TRUE)

ENO = EN;
OUT = IN1 & IN2;

Note: This logical & is not the same as PMAC's. PMAC does a floating point AND. The IEC & is the PMAC & converted to INTeger. See following examples:

PMAC: 4.5 = 4.5 & 5.625
IEC : 4 = 4.5 & 5.25

//**

OR_LD48VAR_INPUT

EN : BOOL;

```

IN1  : REAL48;
IN2  : REAL48;

```

VAR_OUTPUT

```

ENO  : BOOL;
OUT  : REAL48;

```

PROCESS: (Done when EN = TRUE)

```

ENO = EN;
OUT = IN1 | IN2;

```

Note: This logical OR is not the same as PMAC's. PMAC does a floating point OR. The IEC & is the PMAC OR converted to INTegeR. See following examples:

```

PMAC: 5.625 = 4.5 | 5.625
IEC   : 5 = 4.5 | 5.25

```

//**

EOR_LD48

VAR_INPUT

```

EN   : BOOL;
IN1  : REAL48;
IN2  : REAL48;

```

VAR_OUTPUT

```

ENO  : BOOL;
OUT  : REAL48;

```

PROCESS: (Done when EN = TRUE)

```

ENO = EN;
OUT = IN1 ^ IN2;

```

Note: This logical EOR is not the same as PMAC's. PMAC does a floating point EOR. The IEC & is the PMAC EOR converted to INTegeR. See following examples:

```

PMAC: 1.125 = 4.5 ^ 5.625
IEC   : 1 = 4.5 ^ 5.25

```

//**

Complex Arithmetic Functions

SIN_LD48

VAR_INPUT

```

EN   : BOOL;
IN   : REAL48;      In units of Degrees
IN2  : REAL48;

```

VAR_OUTPUT

```

ENO  : BOOL;
OUT  : REAL48;

```


PROCESS: (Done when EN = TRUE)

ENO = EN;
OUT = SIN(IN);

//**

COS_LD48

VAR_INPUT

EN : BOOL;
IN : REAL48; In units of Degrees
IN2 : REAL48;

VAR_OUTPUT

ENO : BOOL;
OUT : REAL48;

PROCESS: (Done when EN = TRUE)

ENO = EN;
OUT = COS(IN);

//**

TAN_LD48

VAR_INPUT

EN : BOOL;
IN : REAL48; In units of Degrees
IN2 : REAL48;

VAR_OUTPUT

ENO : BOOL;
OUT : REAL48;

PROCESS: (Done when EN = TRUE)

ENO = EN;
OUT = TAN(IN);

//**

ASIN_LD48

VAR_INPUT

EN : BOOL;
IN : REAL48;

VAR_OUTPUT

ENO : BOOL;
OUT : REAL48; In units of Degrees

PROCESS: (Done when EN = TRUE)

ENO = EN;
OUT = ASIN(IN);

//**

ACOS_LD48

VAR_INPUT

EN : **BOOL**;**IN** : **REAL48**;

VAR_OUTPUT

ENO : **BOOL**;**OUT** : **REAL48**; In units of DegreesPROCESS: (Done when **EN** = **TRUE**)**ENO** = **EN**;**OUT** = **ACOS**(**IN**);

//*****

ATAN_LD48

VAR_INPUT

EN : **BOOL**;**IN** : **REAL48**;

VAR_OUTPUT

ENO : **BOOL**;**OUT** : **REAL48**; In units of DegreesPROCESS: (Done when **EN** = **TRUE**)**ENO** = **EN**;**OUT** = **ATAN**(**IN**);

//*****

ATAN2_LD48

VAR_INPUT

EN : **BOOL**;**IN1** : **REAL48**;**IN2** : **REAL48**;

VAR_OUTPUT

ENO : **BOOL**;**OUT** : **REAL48**; In units of DegreesPROCESS: (Done when **EN** = **TRUE**)**ENO** = **EN**;**OUT** = **ATAN**(**IN1**/**IN2**);

//*****

SQRT_LD48

VAR_INPUT

EN : **BOOL**;**IN** : **REAL48**;

VAR_OUTPUT

```
    ENO  : BOOL;  
    OUT  : REAL48;
```

```
PROCESS: (Done when EN = TRUE)  
    ENO = EN;  
    If ( IN >= 0 ) // Test for Operation negative  
        OUT = SQRT(IN);  
    Else  
        ENO = FALSE;
```

```
//*****
```

EXP_LD48

```
VAR_INPUT  
    EN   : BOOL;  
    IN   : REAL48;
```

```
VAR_OUTPUT  
    ENO  : BOOL;  
    OUT  : REAL48;
```

```
PROCESS: (Done when EN = TRUE)  
    ENO = EN;  
    OUT = EXP(IN); // e to the IN power ( e**IN)
```

```
//*****
```

LN_LD48

```
VAR_INPUT  
    EN   : BOOL;  
    IN   : REAL48;
```

```
VAR_OUTPUT  
    ENO  : BOOL;  
    OUT  : REAL48;
```

```
PROCESS: (Done when EN = TRUE)  
    ENO = EN;  
    If ( IN >= 0 ) // Test for Operation negative  
        OUT = LN(IN); // Natural Logarithm base e  
    Else  
        ENO = FALSE;
```

```
//*****
```

LOG_LD48

```
VAR_INPUT  
    EN   : BOOL;  
    IN   : REAL48;
```

```
VAR_OUTPUT  
    ENO  : BOOL;  
    OUT  : REAL48;
```

```

PROCESS: (Done when EN = TRUE)
  ENO = EN;
  If ( IN >= 0 ) // Test for Operation negative
    OUT = LOG(IN); // Logarithm base 10
  Else
    ENO = FALSE;

```

```

//*****

```

TRUNC_LD48

```

VAR_INPUT
  EN   : BOOL;
  IN   : REAL48;

```

```

VAR_OUTPUT
  ENO   : BOOL;
  OUT   : REAL48;

```

```

PROCESS: (Done when EN = TRUE)
  ENO = EN;
  OUT = TRUNC( IN );

```

Note: This TRUNC is not the same as PMAC's INT Function. PMAC truncates towards negative infinity. The IEC truncates toward Zero.

Example:

PMAC: -1 = INT(-0.1) ; -2 = INT(-1.1) ; 1 = INT(1.1)

IEC: 0 = TRUNC (-0.1) ; -1 = TRUNC (-1.1) ; 1 = TRUNC (1.1)

```

//*****

```

Limit Functions

MAX_LD24

```

VAR_INPUT
  EN   : BOOL;
  IN1  : INT24;
  IN2  : INT24;

```

```

VAR_OUTPUT
  ENO   : BOOL;
  OUT   : INT24;

```

```

PROCESS: (Done when EN = TRUE)
  ENO = EN;
  If ( IN1 >= IN2 )
    OUT = IN1
  Else
    OUT = IN2

```

```

//*****

```

MIN_LD24

```
VAR_INPUT
  EN   : BOOL;
  IN1  : INT24;
  IN2  : INT24;
```

```
VAR_OUTPUT
  ENO  : BOOL;
  OUT  : INT24;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  If ( IN1 <= IN2 )
    OUT = IN1
  Else
    OUT = IN2
```

```
//************************************************************************
```

MAX_LD48

```
VAR_INPUT
  EN   : BOOL;
  IN1  : REAL48;
  IN2  : REAL48;
```

```
VAR_OUTPUT
  ENO  : BOOL;
  OUT  : REAL48;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  If ( IN1 >= IN2 )
    OUT = IN1
  Else
    OUT = IN2
```

```
//************************************************************************
```

MIN_LD48

```
VAR_INPUT
  EN   : BOOL;
  IN1  : REAL48;
  IN2  : REAL48;
```

```
VAR_OUTPUT
  ENO  : BOOL;
  OUT  : REAL48;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  If ( IN1 <= IN2 )
    OUT = IN1
  Else
    OUT = IN2
```

```
//************************************************************************
```

Assignment Functions

MOVE_LD24

```
VAR_INPUT
  EN   : BOOL;
  IN   : INT24;
```

```
VAR_OUTPUT
  ENO  : BOOL;
  OUT  : INT24;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  OUT = IN;
```

```
//************************************************************************
```

MOVE_LD48

```
VAR_INPUT
  EN   : BOOL;
  IN   : REAL48;
```

```
VAR_OUTPUT
  ENO  : BOOL;
  OUT  : REAL48;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  OUT = IN;
```

```
//************************************************************************
```

Arithmetic Comparison Functions

EQ_ILD24

```
VAR_INPUT
  EN   : BOOL;
  IN1  : INT24;
  IN2  : INT24;
```

```
VAR_OUTPUT
  ENO  : BOOL;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  If( IN1 != IN2 )
    ENO = FALSE;
```

```
/*******
```

GE_ILD24

```
VAR_INPUT
```

```
  EN   : BOOL;
```

```
  IN1  : INT24;
```

```
  IN2  : INT24;
```

```
VAR_OUTPUT
```

```
  ENO  : BOOL;
```

```
PROCESS: (Done when EN = TRUE)
```

```
  ENO = EN;
```

```
  If( IN1 < IN2 )
```

```
    ENO = FALSE;
```

```
/*******
```

GT_ILD24

```
VAR_INPUT
```

```
  EN   : BOOL;
```

```
  IN1  : INT24;
```

```
  IN2  : INT24;
```

```
VAR_OUTPUT
```

```
  ENO  : BOOL;
```

```
PROCESS: (Done when EN = TRUE)
```

```
  ENO = EN;
```

```
  If( IN1 <= IN2 )
```

```
    ENO = FALSE;
```

```
/*******
```

LE_ILD24

```
VAR_INPUT
```

```
  EN   : BOOL;
```

```
  IN1  : INT24;
```

```
  IN2  : INT24;
```

```
VAR_OUTPUT
```

```
  ENO  : BOOL;
```

```
PROCESS: (Done when EN = TRUE)
```

```
  ENO = EN;
```

```
  If( IN1 > IN2 )
```

```
    ENO = FALSE;
```

```
/*******
```

LT_ILD24

VAR_INPUT

EN : BOOL;

IN1 : INT24;

IN2 : INT24;

VAR_OUTPUT

ENO : BOOL;

PROCESS: (Done when EN = TRUE)

ENO = EN;

If(IN1 >= IN2)

ENO = FALSE;

//*****

NE_ILD24

VAR_INPUT

EN : BOOL;

IN1 : INT24;

IN2 : INT24;

VAR_OUTPUT

ENO : BOOL;

PROCESS: (Done when EN = TRUE)

ENO = EN;

If(IN1 == IN2)

ENO = FALSE;

//*****

EQ_ILD48

VAR_INPUT

EN : BOOL;

IN1 : REAL48;

IN2 : REAL48;

VAR_OUTPUT

ENO : BOOL;

PROCESS: (Done when EN = TRUE)

ENO = EN;

If(IN1 != IN2)

ENO = FALSE;

//*****

GE_ILD48

VAR_INPUT

EN : BOOL;

IN1 : REAL48;

IN2 : REAL48;


```
VAR_OUTPUT
  ENO  : BOOL;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  If( IN1 < IN2 )
    ENO = FALSE;
```

```
//*****
```

GT_ILD48

```
VAR_INPUT
  EN    : BOOL;
  IN1   : REAL48;
  IN2   : REAL48;
```

```
VAR_OUTPUT
  ENO  : BOOL;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  If( IN1 <= IN2 )
    ENO = FALSE;
```

```
//*****
```

LE_ILD48

```
VAR_INPUT
  EN    : BOOL;
  IN1   : REAL48;
  IN2   : REAL48;
```

```
VAR_OUTPUT
  ENO  : BOOL;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  If( IN1 > IN2 )
    ENO = FALSE;
```

```
//*****
```

LT_ILD48

```
VAR_INPUT
  EN    : BOOL;
  IN1   : REAL48;
  IN2   : REAL48;
```

```
VAR_OUTPUT
  ENO  : BOOL;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  If( IN1 >= IN2 )
```

```
ENO = FALSE;
```

```
//************************************************************************
```

NE_ILD48

```
VAR_INPUT
```

```
  EN   : BOOL;
```

```
  IN1  : REAL48;
```

```
  IN2  : REAL48;
```

```
VAR_OUTPUT
```

```
  ENO  : BOOL;
```

```
PROCESS: (Done when EN = TRUE)
```

```
  ENO = EN;
```

```
  If( IN1 == IN2 )
```

```
    ENO = FALSE;
```

```
//************************************************************************
```

Data Type Conversion Functions

LD24_TO_LD48

```
VAR_INPUT
```

```
  EN   : BOOL;
```

```
  IN   : INT24;
```

```
VAR_OUTPUT
```

```
  ENO  : BOOL;
```

```
  OUT  : REAL48;
```

```
PROCESS: (Done when EN = TRUE)
```

```
  ENO = EN;
```

```
  OUT = (REAL48) IN; // Convert INT24 to REAL48
```

```
//************************************************************************
```

LD48_TO_LD24

```
VAR_INPUT
```

```
  EN   : BOOL;
```

```
  IN   : REAL48;
```

```
VAR_OUTPUT
```

```
  ENO  : BOOL;
```

```
  OUT  : INT24;
```

```
PROCESS: (Done when EN = TRUE)
```

```
  ENO = EN;
```

```
  OUT = (INT24) IN; // Convert REAL48 to INT24. The lower 24 bit are saved. So if the value is
    Out of range of 24 bits, it may change sign.
```

```
//************************************************************************
```

LD24_TO_INT

```
VAR_INPUT
  EN   : BOOL;
  IN   : INT24;
```

```
VAR_OUTPUT
  ENO  : BOOL;
  OUT  : INT;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  OUT = (INT) IN; // Convert INT24 to INT. The lower 16 bit are saved. So if the value is
                Out of range of 16 bits, it may change sign.
```

//**

INT_TO_LD24

```
VAR_INPUT
  EN   : BOOL;
  IN   : INT;
```

```
VAR_OUTPUT
  ENO  : BOOL;
  OUT  : INT24;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  OUT = (INT24) IN; // Convert INT to INT24. The 16 bit INT is sign extended to 24 bits.
```

//**

GREY4_TO_LD24

```
VAR_INPUT
  EN   : BOOL;
  IN   : INT24; // 4 bit Grey code in lower 4 bits.
```

```
VAR_OUTPUT
  ENO  : BOOL;
  OUT  : INT24;
```

```
PROCESS: (Done when EN = TRUE)
  ENO = EN;
  OUT = (GREY4_to_INT24) IN & 15; // Convert 4 bit INT24 Grey code to INT24. The 4 bit
  grey code is in the lower 4 bits.
```

GREY_IN	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
INT24_OUT	0	1	3	2	6	7	5	4	12	13	15	14	10	11	9	8

//**

Get Time Functions

GetTime_LD48

```

VAR_INPUT
    EN    : BOOL;

VAR_OUTPUT
    ENO   : BOOL;
    OUT   : REAL48;

PROCESS: (Done when EN = TRUE)
    ENO = EN;
    OUT = GetTime(); // Get PMAC Time in milli-seconds since power on.

```

PMAC Ladder LD & SFC Function Blocks

The following are PMAC Function Blocks that allow you to perform special PMAC commands. Since they are Function Blocks, they must be defined as an instance with their own unique label. The normal IEC counter and timer Function Blocks remain as IEC standard TIME and INT data types.

PMAC Ladder Function Blocks	Description
PMAC_CMD_STR	Sends a Command String to PMAC on a rising edge input
PMAC_CMDL_STR	Sends a Command String to PMAC on a level TRUE input
PMAC_DISP_VAR	Sends a formatted floating variable to PMAC's Display buffer
PMAC_DISP_STR	Sends a String to PMAC's Display buffer
PMAC_LOCK	Will LOCK one of the 8 LOCK bits if it is not LOCKed
PMAC_UNLOCK	Will UNLOCK one of the 8 LOCK bits
PMAC_SET_PHASE	Will set the Motor PHASE value = Ixx75 for the selected Motors
PMAC_CYCLE_TIME	Will compute the Current and Maximum PMAC background Cycle Time

Special PMAC Function Blocks

```

//*****

```

PMAC_CMD_STR

```

VAR_INPUT
    ENA    : BOOL R_EDGE;    // Enable on rising edge
    TYP    : PMAC_CMD_TYP;    // Type of CMD and PMAC PORT
    STR    : STRING;          // String

VAR_OUTPUT
    OK: BOOL;                // Command String was sent to PMAC

PROCESS:
    If ( (I5 == 2) && (PLC Command not in process) && (ENA on rising edge))
        Then send STR to PMAC PLC CMD buffer with the following CMD_TYP:
            OK = TRUE
        endif

```

```
TYPE PMAC_CMD_TYP :
  ( DUMMY0,      (* 0 *)
    SENDP,      (* 1 Send to Parallel Port *)
    SENDR,      (* 2 Send to Dual Port Ram *)
    SENDS,      (* 3 Send to Serial Port *)
    SENDA,      (* 4 Send to Auxiliary Serial Port *)
    DUMMY1,      (* 5 *)
    DUMMY2,      (* 6 *)
    DUMMY3,      (* 7 *)
    CMD,         (* 8 CMD to No Port *)
    CMDP,        (* 9  CMD to Parallel Port *)
    CMDR,        (* 10 CMD to Dual Port Ram *)
    CMDS,        (* 11 CMD to Serial Port *)
    CMDA         (* 12 CMD to Auxiliary Serial Port*)
  );
END_TYPE

//*****

PMAC_CMDL_STR
VAR_INPUT
  ENA   : BOOL;      // Enable on level = TRUE
  TYP   : PMAC_CMD_TYP;  // Type of CMD and PMAC PORT
  STR   : STRING;     // String

VAR_OUTPUT
  OK: BOOL;          // Command String was sent to PMAC

PROCESS:
  If ( (I5 == 2) && (PLC Command not in process) && (ENA == TRUE))
    Then send STR to PMAC PLC CMD buffer with the following CMD_TYP:
      OK = TRUE
    endif

TYPE PMAC_CMD_TYP:
  ( DUMMY0,      (* 0 *)
    SENDP,      (* 1 Send to Parallel Port *)
    SENDR,      (* 2 Send to Dual Port Ram *)
    SENDS,      (* 3 Send to Serial Port *)
    SENDA,      (* 4 Send to Auxiliary Serial Port *)
    DUMMY1,      (* 5 *)
    DUMMY2,      (* 6 *)
    DUMMY3,      (* 7 *)
    CMD,         (* 8 CMD to No Port *)
    CMDP,        (* 9  CMD to Parallel Port *)
    CMDR,        (* 10 CMD to Dual Port Ram *)
    CMDS,        (* 11 CMD to Serial Port *)
    CMDA         (* 12 CMD to Auxiliary Serial Port*)
  );
END_TYPE

//*****
```

PMAC_DISP_VAR

VAR_INPUT

ENA : **BOOL**; // Enable on level = TRUE
START : **INT24**; // Display Buffer start location (Range 0 – 79)
FVAR : **REAL48**; // Displayed floating point variable
PREC : **INT24**; // Displayed variable width (Range 2 – 16)
DECPT : **INT24**; // Displayed variable number of decimal places (Range 0 – 9)

VAR_OUTPUT

OK: **BOOL**; // Formatted FVAR was sent to PMAC display buffer

PROCESS:

if(ENA && (START <80) && (START >= 0) && (PREC < 17) && (PREC > 1) && (DECPT < 10)
 && (DECPT >= 0) && (PREC >= (DECPT+2)))
 Then send formatted FVAR to PMAC display buffer.
OK = TRUE
 endif

//**

PMAC_DISP_STR

VAR_INPUT

ENA : **BOOL** **R_EDGE**; // Enable on rising edge
START : **INT24**; // Display Buffer start location (Range 0 – 79)
STR : **STRING**; // Display String

VAR_OUTPUT

OK: **BOOL**; // String was sent to PMAC display buffer

PROCESS:

if(ENA && (START <80) && (START >= 0)
 Then send STRing to PMAC display buffer.
OK = TRUE
 endif

//**

PMAC_LOCK

VAR_INPUT

ENA : **BOOL**; // Enable on level TRUE
NUM : **INT24**; // Lock Number (Range 0 – 7)

VAR_OUTPUT

OK: **BOOL**; // LOCK occurred

PROCESS:

if(ENA && (NUM < 8) && (NUM >= 0)&& LOCK(NUM) != LOCK))
 LOCK(NUM) = LOCK
OK = TRUE
 endif

//**

PMAC_UNLOCK

```
VAR_INPUT
  ENA   : BOOL;           // Enable on level TRUE
  NUM   : INT24;          // Lock Number (Range 0 – 7)
```

```
VAR_OUTPUT
  OK: BOOL;               // UNLOCK occurred
```

```
PROCESS:
  if( ENA && (NUM < 8) && (NUM >= 0) )
    LOCK(NUM) = UNLOCK
    OK = TRUE
  endif
```

PMAC_SET_PHASE

```
VAR_INPUT
  ENA   : BOOL R_EDGE; // Enable on rising edge
  MTR_MASK : DWORD;    // MTR Mask for 32 motors (B0 = MTR #1)
```

```
VAR_OUTPUT
  OK: BOOL;             // MTR Phase was initialized
```

```
PROCESS:
  if( ENA on rising edge )
    For MTR bits of MTR_MASK
      Set MTR PHASE = Ixx75
    OK = TRUE
  endif
```

PMAC_CYCLE_TIME

```
VAR_INPUT
  ENA   : BOOL;           // Enable on level TRUE
```

```
VAR_OUTPUT
  TM   : REAL48;          // PMAC Current Cycle Time in milli-seconds
  TMAX : REAL48;          // PMAC Maximum Cycle Time in milli-seconds
  SF   : REAL48;          // PMAC Cycle Time Scale Factor
```

```
PROCESS:
  if( ENA == TRUE )
    TM = SF * X:$22; // PMAC Current Cycle Time in milli-seconds
    TMAX = SF * Y:$22; // PMAC Maximum Cycle Time in milli-seconds
  Else
    SF = 0.000203458/( I52 + 1.0 )
    Y:$22 = X:$22; // Set Maximum = Current PMAC Cycle Time
  endif
```

Note: This is PMAC background cycle time, which is also PMAC Ladder's cycle time. It is time period that these processes are executed. Taking 1/TM would give the current number of CYCLES per second.

//**

IEC LADDER & SFC FUNCTION BLOCKS

The following are the standard IEC Function Blocks that normally included in Ladder language.

IEC Ladder Function Blocks	Description
F_TRIG	Falling Edge Trigger
R_TRIG	Rising Edge Trigger
RS	RS – Flip – Flop
SR	SR – Flip – Flop
TOF	Delayed Timer Off Trigger
TON	Delayed Timer On Trigger
TP	Timer Pulse
CTD	Counter Down Count
CTU	Counter Up Count
CTUD	Counter Up and Down Count

Standard IEC Function Blocks

F_TRIG

```
VAR_INPUT
    CLK : BOOL;
```

```
VAR_OUTPUT
    Q : BOOL;
```

PROCESS: // The state of Q is "1" at the output only during the one cycle where the change of state of "CLK" was a falling edge signal.

//**

R_TRIG

```
VAR_INPUT
    CLK : BOOL;
```

```
VAR_OUTPUT
    Q : BOOL;
```

PROCESS: // The state of Q is "1" at the output only during the one cycle where the change of state of "CLK" was a falling edge signal.

//**

RS

```
VAR_INPUT
```



```
SET    : BOOL;  
RESET1 : BOOL;
```

```
VAR_OUTPUT  
Q1      : BOOL;
```

Process:

RS implements a Flip-Flop. Output Q1 delivers the current state of that Flip-Flop. The initial value of the Flip-Flop is FALSE. TRUE on input SET will set the Flip-Flop, TRUE on input RESET1 will reset the Flip-Flop to FALSE. If both SET and RESET1 are TRUE in one call, RS is reset dominant, which means the RESET will override the SET and the Flip-Flop will hence be reset.

```
//*****
```

SR

```
VAR_INPUT  
SET1    : BOOL;  
RESET   : BOOL;
```

```
VAR_OUTPUT  
Q1      : BOOL;
```

Process

SR implements a Flip-Flop. Output Q1 delivers the current state of that Flip-Flop. The initial value of the Flip-Flop is FALSE. TRUE on input SET will set the Flip-Flop, TRUE on input RESET1 will reset the Flip-Flop to FALSE. If both SET and RESET1 are TRUE in one call, SR is set dominant, which means the SET will override the RESET and the Flip-Flop will hence be set.

```
//*****
```

TOF

```
VAR_INPUT  
IN     : BOOL;  
PT     : TIME;
```

```
VAR_OUTPUT  
Q       : BOOL;  
ET      : TIME;
```

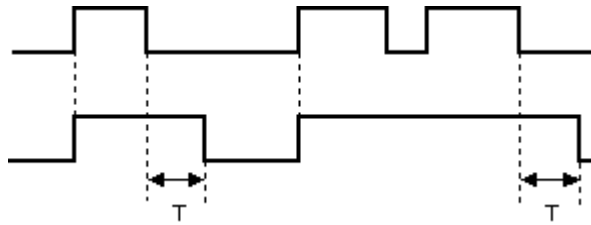
Process:

If the state of the input operand "IN" is "1", this will be passed to the output operand "Q" without any delay. If there is a falling edge, a timer function will be started lasting as long an interval as specified by the operand "PT"

If the timer is running, a change of state at the input "IN" to "0" will have no implications. It is after the time is up that the operand "Q" will change to the state "0". If the "PT" value changes after the start, it will have no implications until there is the next rising edge of the operand "IN".

The operand "ET" contains the current timer value. If the time is up, the operand "ET" will keep its value as long as the operand "IN" has the value "1". If the state of the "IN" operand changes to "0", the value of "ET" will switch to "0".

If the input "IN" is switched off, this will switch off the output "Q" after an interval specified by the delay value.



Time Diagram:

```

//*****

```

TON

```

VAR_INPUT
    IN  : BOOL;
    PT  : TIME;

VAR_OUTPUT
    Q    : BOOL;
    ET   : TIME;

```

Process:

The rising edge of the input operand **IN** will start the timer **TON**, and it will run as long a time interval as specified by the operand **PT**.

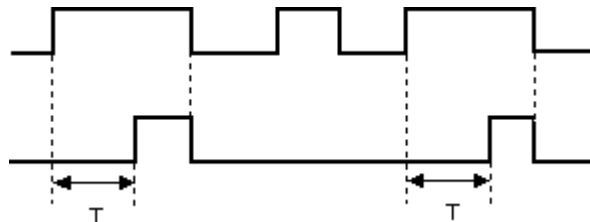
While the timer is running, the output operand **Q** will have the value **0**. If the time is up, the state will change to **1** and keep this value until the operand **IN** changes to **0**.

If the **PT** value changes after the timer has been started, this will have no implications until the next rising edge of the operand **IN**.

The output operand **ET** contains the current timer value. If the time is up, the operand **ET** will keep its value as long as the operand **IN** has the value **1**. If the state of the **IN** operand changes to **0**, the value of **ET** will switch to **0**.

If the input "IN" is switched on, this will switch on the output **Q** after an interval specified by the delay value.

Time diagram:



```

//*****

```

TP

```

VAR_INPUT
    IN  : BOOL;
    PT  : TIME;

VAR_OUTPUT

```

```

Q      : BOOL;
ET     : TIME;

```

Process:

A rising edge of the input operand **IN** will start the timing function of the timer **TP**, and it will run as long an interval as specified by the operand **PT**.

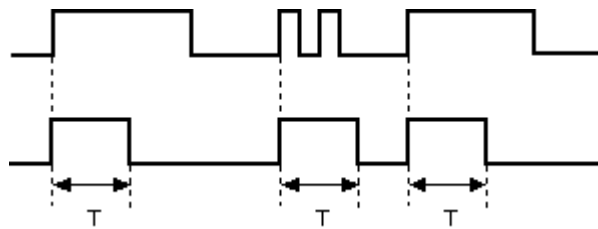
While the timer is running, the output operand **Q** will have the state **1**. Any changes of state at the input **IN** will have no implications on the procedure.

If the **PT** value changes after the start, this will not have any implications before the next rising edge of the **IN** operand.

The output operand **ET** contains the current timer value. If the operand **IN** has the state **1** after the time is up, the operand **ET** will keep its value.

Every edge occurring while the timer is not running will cause an impulse that lasts as long as specified by **PT**.

time diagram:



```

//*****

```

CTD

```

VAR_INPUT
  IN      : BOOL R_EDGE;
  LOAD    : BOOL ;
  PV      : INT;

```

```

VAR_OUTPUT
  Q       : BOOL;
  CV      : INT;

```

Process:

The function block **CTD** serves for counting down impulses received from the input operand **CD**. On initialization, the counter will be set to **0**.

If the operand **LOAD** is **1**, the value received by the operand **PV** will be taken over as a value into the counter.

Each rising edge at the input **CD** will decrease the counter by **1**.

The output operand **CV** contains the current value of the counter. If the counter value is > 0 , the output operand **Q** will have the Boolean value **0**. If the counter value reaches zero, the output **Q** will be set to **1**.

```

//*****

```

CTU

```

VAR_INPUT
  IN      : BOOL R_EDGE;

```

```

RESET : BOOL ;
LOAD  : BOOL ;
PV    : INT;

```

```

VAR_OUTPUT
  Q      : BOOL;
  CV     : INT;

```

Process:

The function block **CTD** serves for counting up impulses received from the input operand **CD**. On initialization, the counter will be set to **0**.

The counter value will be reset if the operand **RESET** receives the value **1**.

Each rising edge at the input **CD** will increase the counter by **1**.

The output operand **CV** contains the current value of the counter. If the counter value is below the margin value **PV**, the output operand **Q** will have the Boolean value **0**. If the counter value equals the PV value, the output **Q** will be set to **1**.

```

//*****

```

CTUD

```

VAR_INPUT
  CU      : BOOL R_EDGE;
  CD      : BOOL R_EDGE;
  RESET : BOOL ;
  LOAD : BOOL ;
  PV     : INT;

```

```

VAR_OUTPUT
  QU      : BOOL;
  QD      : BOOL;
  CV     : INT;

```

Process:

The function block **CTUD** serves for counting up and down impulses. On initializations, the counter will be set to the value **0**. Every rising edge at the input operand **CU** will increase the counter by **1**, while every rising edge at the input **CD** will decrease it by **1**.

If the operand **LOAD** is **1**, the value received by the operand **PV** will be taken over as a value into the counter.

The counter value will be reset to zero if the operand **RESET** receives the value **1**. While the static state of the operand **RESET** remains unchanged, the counting conditions or the load condition will have no implication, independent of their value.

The output operand **CV** contains the current value of the counter. If the counter value is below the margin value **PV** and not Zero, the output operand **QU** or **QD** will have the Boolean value **0**. If the counter value reaches the PV value, the output **QU** will be set to **1**. If the counter value reaches zero, the output **QD** will be set to **1**.

PMAC Ladder SFC Instruction List (IL) Functions

PMAC SFC IL Functions	Returned Data Type	Description
Arithmetic		Standard Arithmetic Functions
ABS_IL24	INT24	PMAC 24 bit Integer ABS (Absolute Value)
ADD_IL24	INT24	PMAC 24 bit Integer ADD
SUB_IL24	INT24	PMAC 24 bit Integer SUB
MUL_IL24	INT24	PMAC 24 bit Integer MUL
DIV_IL24	INT24	PMAC 24 bit Integer DIV
MOD_IL24	INT24	PMAC 24 bit Integer MOD (Modulo)
NEG_IL24	INT24	PMAC 24 bit Integer NEG (Negate)
ABS_IL48	REAL48	PMAC 48 bit Floating Point ABS
ADD_IL48	REAL48	PMAC 48 bit Floating Point ADD
SUB_IL48	REAL48	PMAC 48 bit Floating Point SUB
MUL_IL48	REAL48	PMAC 48 bit Floating Point MUL
DIV_IL48	REAL48	PMAC 48 bit Floating Point DIV
MOD_IL48	REAL48	PMAC 48 bit Floating Point MOD
NEG_IL48	REAL48	PMAC 48 bit Floating Point NEG
Logical		Standard Logical Functions
AND_IL24	INT24	PMAC 24 bit Integer AND
OR_IL24	INT24	PMAC 24 bit Integer OR
EOR_IL24	INT24	PMAC 24 bit Integer EOR (Exclusive OR)
NOT_IL24	INT24	PMAC 24 bit Integer NOT (I'S Compliment)
ROL_IL24	INT24	PMAC 24 bit Unsigned Integer Logical Rotate Left
ROR_IL24	INT24	PMAC 24 bit Unsigned Integer Logical Rotate Right
SHL_IL24	INT24	PMAC 24 bit Unsigned Integer Logical Shift Left
SHR_IL24	INT24	PMAC 24 bit Unsigned Integer Logical Shift Right
AND_IL48	REAL48	PMAC 48 bit Floating Point Logical AND
OR_IL48	REAL48	PMAC 48 bit Floating Point Logical OR
EOR_IL48	REAL48	PMAC 48 bit Floating Point Logical EOR
Arithmetic		Complex Arithmetic Functions
SIN_IL48	REAL48	PMAC 48 bit Floating Point SIN (SIN value in Degrees)
COS_IL48	REAL48	PMAC 48 bit Floating Point COS (COS value in Degrees)
TAN_IL48	REAL48	PMAC 48 bit Floating Point TAN (TAN value in Degrees)

ASIN_IL48	REAL48	PMAC 48 bit Floating Point ASIN (ASIN return in Degrees)
ACOS_IL48	REAL48	PMAC 48 bit Floating Point ACOS (ACOS return in Degrees)
ATAN_IL48	REAL48	PMAC 48 bit Floating Point ATAN (ATAN return in Degrees)
ATAN2_IL48	REAL48	PMAC 48 bit Floating Point ATAN2 (2 Input ATAN return in Degrees)
SQRT_IL48	REAL48	PMAC 48 bit Floating Point SQRT (Square Root)
EXP_IL48	REAL48	PMAC 48 bit Floating Point EXP (e to the X)
LN_IL48	REAL48	PMAC 48 bit Floating Point LN (Natural LOG base e)
LOG_IL48	REAL48	PMAC 48 bit Floating Point LOG (LOG base 10)
TRUNC_IL48	REAL48	PMAC 48 bit Floating Point Truncate down to the nearest Integer Value
Limit Functions		Limit Functions
MAX_IL24	INT24	PMAC 24 bit Integer MAX (Maximum Value)
MIN_IL24	INT24	PMAC 24 bit Integer MIN (Minimum Value)
MAX_IL48	REAL48	PMAC 48 bit Floating Point MAX (Maximum Value)
MIN_IL48	REAL48	PMAC 48 bit Floating Point MIN (Minimum Value)
Comparison		Arithmetic Comparison Functions
EQ_ILD24	INT24	PMAC 24 bit Integer EQ (Equal Compare)
GE_ILD24	INT24	PMAC 24 bit Integer GE (Greater Than or Equal Compare)
GT_ILD24	INT24	PMAC 24 bit Integer GT (Greater Than Compare)
LE_ILD24	INT24	PMAC 24 bit Integer LE (Less Than or Equal Compare)
LT_ILD24	INT24	PMAC 24 bit Integer LT (Less Than Compare)
NE_ILD24	INT24	PMAC 24 bit Integer NE (Not Equal Compare)
EQ_ILD48	REAL48	PMAC 48 bit Floating Point EQ (Equal Compare)
GE_ILD48	REAL48	PMAC 48 bit Floating Point GE (Greater Than or Equal Compare)
GT_ILD48	REAL48	PMAC 48 bit Floating Point GT (Greater Than Compare)
LE_ILD48	REAL48	PMAC 48 bit Floating Point LE (Less Than or Equal Compare)
LT_ILD48	REAL48	PMAC 48 bit Floating Point LT (Less Than Compare)
NE_ILD48	REAL48	PMAC 48 bit Floating Point NE (Not Equal Compare)
Conversions		Data Type Conversion Functions
IL24_TO_IL48	INT24/REAL48	PMAC 24 bit Integer TO 48 bit Floating Point
IL48_TO_IL24	REAL48/INT24	PMAC 48 bit Floating Point TO 24 bit Integer
IL24_TO_INT	INT24/INT	PMAC 24 bit Integer TO 16 bit Integer
INT_TO_IL24	INT/INT24	PMAC 16 bit Integer TO 24 bit Integer
GREY4_TO_IL24	GREY4/ INT24	PMAC 24 bit Integer 4 bit Grey Code TO 24 bit Integer
Time		Time Functions
GetTime_IL48	REAL48	PMAC 48 bit Floating Point Get Time (milli-seconds) since Power On

These are functions so they will return (**RTN**) a value of a certain Data Type when CALLED.

Standard Arithmetic Functions

ABS_IL24 : INT24

VAR_INPUT

IN : INT24;

Process:

RTN = ABS(IN);

//**

ADD_IL24 : INT24

VAR_INPUT

IN1 : INT24;

IN2 : INT24;

Process:

RTN = IN1 + IN2; // If an overflow occurs, the lower 24 bits are returned.

//**

SUB_IL24 : INT24

VAR_INPUT

IN1 : INT24;

IN2 : INT24;

Process:

RTN = IN1 - IN2; // If an overflow occurs, the lower 24 bits are returned.

//**

MUL_IL24 : INT24

VAR_INPUT

IN1 : INT24;

IN2 : INT24;

Process:

RTN = IN1 * IN2; // If an overflow occurs, the lower 24 bits are returned.

//**

DIV_IL24 : INT24

VAR_INPUT

IN1 : INT24;

IN2 : INT24;

Process:

If (IN2 != 0) // Test for divide by zero

RTN = IN1 / IN2;

If (RTN == 8388608)

OUT = 8388607 // Max 24 bit positive value

//**

MOD_IL24 : INT24

VAR_INPUT

IN1 : INT24;

IN2 : INT24;

Process:

If (IN2 != 0) // Test for divide by zero
RTN = IN1 % IN2;

//***

NEG_IL24 : INT24

VAR_INPUT

IN : INT24;

Process:

RTN = -IN;

//***

ABS_IL48 : REAL48

VAR_INPUT

IN : REAL48;

Process:

RTN = ABS(IN);

//***

ADD_IL48 : REAL48

VAR_INPUT

IN1 : REAL48;

IN2 : REAL48;

Process:

RTN = IN1 + IN2;

//***

SUB_IL48 : REAL48

VAR_INPUT

IN1 : REAL48;

IN2 : REAL48;

Process:

RTN = IN1 - IN2;

//***

MUL_IL48 : REAL48

VAR_INPUT

IN1 : REAL48;

IN2 : REAL48;

Process:

RTN = IN1 * IN2;

//**

DIV_IL48 : REAL48

VAR_INPUT

IN1 : REAL48;

IN2 : REAL48;

Process:

If (IN2 != 0) // Test for Operation overflow

RTN = IN1 / IN2;

//**

MOD_IL48 : REAL48

VAR_INPUT

IN1 : REAL48;

IN2 : REAL48;

Process:

If (IN2 != 0) // Test for Operation overflow

RTN = IN1 % IN2;

//**

NEG_IL48 : REAL48

VAR_INPUT

IN : REAL48;

Process:

RTN = - IN;

//**

Standard Logical Functions

AND_IL24 : INT24

VAR_INPUT

IN1 : INT24;

IN2 : INT24;

Process:

RTN = IN1 & IN2;

//**

OR_IL24 : INT24

VAR_INPUT

```

IN1  : INT24;
IN2  : INT24;

```

```

Process:
RTN = IN1 | IN2;

```

```

//*****

```

EOR_IL24 : INT24

```

VAR_INPUT
IN1  : INT24;
IN2  : INT24;

```

```

Process:
RTN = IN1 ^ IN2;

```

```

//*****

```

NOT_IL24 : INT24

```

VAR_INPUT
IN   : INT24;

```

```

Process:
RTN = ~IN; // The One's compliment ( 0's converted to 1' and 1's converted to 0's)

```

```

//*****

```

ROL_IL24 : INT24

```

VAR_INPUT
IN   : INT24; // Input
N    : INT24; // Rotate amount

```

```

Process:
RTN = ROL(IN); // 24 bit word Rotate Left ( MSB goes to LSB)

```

```

//*****

```

ROR_IL24 : INT24

```

VAR_INPUT
IN   : INT24; // Input
N    : INT24; // Rotate amount

```

```

Process:
RTN = ROR(IN); // 24 bit word Rotate Right ( LSB goes to MSB )

```

```

//*****

```

SHL_IL24 : INT24

```

VAR_INPUT
IN   : INT24; // Input
N    : INT24; // Rotate amount

```

Process:

RTN = SHL(IN); // 24 bit word Logical Shift Left – Zero fills lower bits

//**

SHR_IL24 : INT24

VAR_INPUT

IN : INT24; // Input

N : INT24; // Rotate amount

Process:

RTN = ROR(IN); // 24 bit word Logical Shift Right – Zero fills upper bits

//**

AND_IL48 : REAL48

VAR_INPUT

IN1 : REAL48;

IN2 : REAL48;

Process:

RTN = IN1 & IN2;

Note: This logical & is not the same as PMAC's. PMAC does a floating point AND. The IEC & is the PMAC & converted to INTeger. See following examples:

PMAC: 4.5 = 4.5 & 5.625

IEC : 4 = 4.5 & 5.25

//**

OR_IL48 : REAL48

VAR_INPUT

IN1 : REAL48;

IN2 : REAL48;

Process:

RTN = IN1 | IN2;

Note: This logical OR is not the same as PMAC's. PMAC does a floating point OR. The IEC & is the PMAC OR converted to INTeger. See following examples:

PMAC: 5.625 = 4.5 | 5.625

IEC : 5 = 4.5 | 5.25

//**

EOR_IL48 : REAL48

VAR_INPUT

IN1 : REAL48;

IN2 : REAL48;

Process:

RTN = IN1 ^ IN2;

Note: This logical EOR is not the same as PMAC's. PMAC does a floating point EOR. The IEC & is the PMAC EOR converted to INTegeR. See following examples:

PMAC: $1.125 = 4.5 \wedge 5.625$

IEC : $1 = 4.5 \wedge 5.25$

//**

Complex Arithmetic Functions

SIN_IL48 : REAL48

VAR_INPUT

IN : REAL48; In units of Degrees

Process:

RTN = SIN(IN);

//**

COS_IL48 : REAL48

VAR_INPUT

IN : REAL48; In units of Degrees

Process:

RTN = COS(IN);

//**

TAN_IL48: REAL48

VAR_INPUT

IN : REAL48; In units of Degrees

Process:

RTN = TAN(IN);

//**

ASIN_IL48 : REAL48

VAR_INPUT

IN : REAL48;

Process:

RTN = ASIN(IN);

//**

ACOS_IL48 : REAL48

VAR_INPUT
IN : REAL48;

Process:
RTN = ACOS(IN);

//**

ATAN_IL48 : REAL48

VAR_INPUT
IN : REAL48;

Process:
RTN = ATAN(IN);

//**

ATAN2_IL48 : REAL48

VAR_INPUT
IN1 : REAL48;
IN2 : REAL48;

Process:
RTN = ATAN(IN1/IN2);

//**

SQRT_IL48 : REAL48

VAR_INPUT
EN : BOOL;

Process:
If (IN >= 0) // Test for Operation negative
RTN = SQRT(IN);

//**

EXP_IL48 : REAL48

VAR_INPUT
IN : REAL48;

Process:
RTN = EXP(IN); // e to the IN power (e**IN)

//**

LN_IL48 : REAL48

VAR_INPUT
IN : REAL48;

Process:

If (IN >= 0) // Test for Operation negative
RTN = LN(IN); // Natural Logarithm base e

//**

LOG_IL48 : REAL48

VAR_INPUT
 EN : BOOL;
 IN : REAL48;

Process:

ENO = EN;
If (IN >= 0) // Test for Operation negative
RTN = LOG(IN); // Logarithm base 10

//**

TRUNC_IL48 : REAL48

VAR_INPUT
 IN : REAL48;

Process:

RTN = TRUNC(IN);

Note: This TRUNC is not the same as PMAC's INT Function. PMAC truncates towards negative infinity. The IEC truncates toward Zero.

Example:

PMAC: -1 = INT(-0.1) ; -2 = INT(-1.1) ; 1 = INT(1.1)

IEC: 0 = TRUNC (-0.1) ; -1 = TRUNC (-1.1) ; 1 = TRUNC (1.1)

//**

Limit Functions

MAX_IL24 : INT24

VAR_INPUT
 IN1 : INT24;
 IN2 : INT24;

Process:

If (IN1 >= IN2)
RTN = IN1
Else
RTN = IN2

//**

MIN_IL24 : INT24

VAR_INPUT
 IN1 : INT24;
 IN2 : INT24;

```
Process:
  If ( IN1 <= IN2 )
    RTN = IN1
  Else
    RTN = IN2
```

```
//*****
```

MAX_IL48 : REAL48

```
VAR_INPUT
  IN1  : REAL48;
  IN2  : REAL48;
```

```
Process:
  If ( IN1 >= IN2 )
    RTN = IN1
  Else
    RTN = IN2
```

```
//*****
```

MIN_IL48 : REAL48

```
VAR_INPUT
  IN1  : REAL48;
  IN2  : REAL48;
```

```
Process:
  If ( IN1 <= IN2 )
    RTN = IN1
  Else
    RTN = IN2
```

```
//*****
```

Arithmetic Comparison Functions

EQ_ILD24 : BOOL

```
VAR_INPUT
  IN1  : INT24;
  IN2  : INT24;
```

```
Process:
  If( IN1 != IN2 )
    RTN = FALSE;
  Else
    RTN = TRUE
```

```
//*****
```

GE_ILD24 : BOOL

```
VAR_INPUT
```

```
IN1  : INT24;  
IN2  : INT24;
```

Process:

```
  If( IN1 < IN2 )  
    RTN = FALSE;  
  Else  
    RTN = TRUE
```

```
//*****
```

GT_ILD24 : BOOL

```
VAR_INPUT
```

```
IN1  : INT24;  
IN2  : INT24;
```

Process:

```
  If( IN1 <= IN2 )  
    RTN = FALSE;  
  Else  
    RTN = TRUE
```

```
//*****
```

LE_ILD24 : BOOL

```
VAR_INPUT
```

```
IN1  : INT24;  
IN2  : INT24;
```

Process:

```
  If( IN1 > IN2 )  
    RTN = FALSE;  
  Else  
    RTN = TRUE
```

```
//*****
```

LT_ILD24 : BOOL

```
VAR_INPUT
```

```
IN1  : INT24;  
IN2  : INT24;
```

Process:

```
  If( IN1 >= IN2 )  
    RTN = FALSE;  
  Else  
    RTN = TRUE
```

```
//*****
```


NE_ILD24 : BOOL

VAR_INPUT
 IN1 : INT24;
 IN2 : INT24;

Process:

 If(IN1 == IN2)
 RTN = FALSE;
 Else
 RTN = TRUE

//*****

EQ_ILD48 : BOOL

VAR_INPUT
 IN1 : REAL48;
 IN2 : REAL48;

Process:

 If(IN1 != IN2)
 RTN = FALSE;
 Else
 RTN = TRUE

//*****

GE_ILD48 : BOOL

VAR_INPUT
 IN1 : REAL48;
 IN2 : REAL48;

Process:

 If(IN1 < IN2)
 RTN = FALSE;
 Else
 RTN = TRUE

//*****

GT_ILD48 : BOOL

VAR_INPUT
 IN1 : REAL48;
 IN2 : REAL48;

Process:

 If(IN1 <= IN2)
 RTN = FALSE;
 Else
 RTN = TRUE

//*****

LE_ILD48 : BOOL

```
VAR_INPUT
  IN1  : REAL48;
  IN2  : REAL48;
```

Process:

```
If( IN1 > IN2 )
  RTN = FALSE;
Else
  RTN = TRUE
```

```
//************************************************************************
```

LT_ILD48 : BOOL

```
VAR_INPUT
  IN1  : REAL48;
  IN2  : REAL48;
```

Process:

```
If( IN1 >= IN2 )
  RTN = FALSE;
Else
  RTN = TRUE
```

```
//************************************************************************
```

NE_ILD48 : BOOL

```
VAR_INPUT
I    N1    : REAL48;
     IN2    : REAL48;
```

Process:

```
If( IN1 == IN2 )
  RTN = FALSE;
Else
  RTN = TRUE
```

```
//************************************************************************
```

Data Type Conversion Functions

LD24_TO_IL48 : REAL48

```
VAR_INPUT
  IN  : INT24;
```

Process:

```
RTN = (REAL48) IN; // Convert INT24 to REAL48
```

```
//************************************************************************
```

LD48_TO_IL24 : INT24

VAR_INPUT
IN : REAL48;

Process:

RTN = (INT24) IN; // Convert REAL48 to INT24. The lower 24 bit are saved. So if the value is
Out of range of 24 bits, it may change sign.

//**

LD24_TO_INT : INT

VAR_INPUT
IN : INT24;

Process:

RTN = (INT) IN; // Convert INT24 to INT. The lower 16 bit are saved. So if the value is
Out of range of 16 bits, it may change sign.

//**

INT_TO_LD24 : INT24

VAR_INPUT
IN : INT;

Process:

RTN = (INT24) IN; // Convert INT to INT24. The 16 bit INT is sign extended to 24 bits.

//**

GREY4_TO_LD24 : INT24

VAR_INPUT
IN : INT24; // 4 bit Grey code in lower 4 bits.

Process:

RTN = (GREY4_to_INT24) IN & 15; // Convert 4 bit INT24 Grey code to INT24. The 4 bit
grey code is in the lower 4 bits.

GREY _IN	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
INT24 _OUT	0	1	3	2	6	7	5	4	12	13	15	14	10	11	9	8

//**

Get Time Functions

GetTime_IL48 : REAL48

VAR_INPUT
 EN : **BOOL**;

Process:

ENO = **EN**;

RTN = **GetTime()**; // Get PMAC Time in milli-seconds since power on.

//*****

PMAC Ladder Input & Output

In the IEC environment accessing physical or logical inputs, output or memory of a PLC (TURBO PMAC in this case) begins with the keyword **AT**. The physical or logical location begins with the key character **%**. Following the percent character is the character that determines whether it is an input (**I**), output (**Q**) or an input and output (**M**). After the **%I**, **%Q**, **%M** it is up to the particular PLC as to how to define the address. However, as a rule in separating fields in the address, the decimal point character (.) is used and that is the case here.

Example of direct Input and Output:

```
VAR
    MyFloat1      AT %IP5 : REAL48;
                  (* Inputs from PMAC P5 into MyFloat1 *)
    MyFloat2      AT %QM6 : REAL48;
                  (* Outputs to PMAC M6 from MyFloat2 *)
    MyFloat3      AT %MQ7 : REAL48;
                  (* Inputs from PMAC Q7 into MyFloat3 at begin of scan*)
                  (* Then Outputs to PMAC Q7 from MyFloat3 at end of scan*)
END_VAR
```

IEC-1131 Input - Output Qualifier

Follows % character

Character	Process	Comment
I	Input to Process Image	Input only
Q	Output from Process Image	Output only
M	Input to Process Image & Output from Process Image	Input & Output

PMAC Address Qualifiers

Follows “%I, %Q, %M” characters

Character String	Syntax	Range	PMAC Data Type	Example
I	I<n>	n=0 – 8191	I variable REAL48	“AT%I800” “AT%QI5” “AT%MI130”
M	M<n>	n=0 – 8191	M variable REAL48	“AT%IM800” “AT%QM5” “AT%MM130”
P	P<n>	n=0 – 8191	P variable REAL48	“AT%IP800” “AT%QP5” “AT%MP130”
Q	Q<n>	n=0 – 8191	Q variable REAL48	“AT%IQ800” “AT%QQ5” “AT%MQ130”
L	L<m>	m=0 – 0x7FFFF (mem. Addr.)	Direct Memory REAL48	“AT%IL800” “AT%QL5” “AT%ML130”
D	D<m>	m=0 – 0x7FFFFFF	Direct Long 48 bit	“AT%ID800”

			integer	“AT%QD5” “AT%MD0x3C”
DP	DP<m>	m=0 – 0x7FFFF	Direct DPR 32 bit integer	“AT%IDP 0x60000” “AT%QDP 0x60000” “AT%MDP 0x60000”
F	F<m>	m=0-0x7FFFF	Direct DPR 32bit Float to REAL48	“AT%IF0x60000” “AT%QF0x60001” “AT%MF0x60002”
X	X<addr.offset.width.sign> (X: direct memory)	addr=0 – 0x7FFFFF offset=0-23, width=1,4,8,12,16,20,24 Sign =0/1 U/S	Direct INT24	“AT%IX123.22” “AT%QX123.0” “AT%MX0x122.24”
Y	Y<addr.offset.width.sign> (Y: direct memory)	Addr=0 – 0x7FFFFF Offset=0-23, width=1,4,8,12,16,20,24 sign =0/1 U/S	Direct INT24	“AT%IY123.22” “AT%QY123.0” “AT%MY122.24”
MM	MM<I/M/PQ m.n>	m=0-15 (Mst.#) n=0-8191	MACRO MstToMst Read from or Write to I/M/P/Q variable REAL48	“AT%IMMI1.5” “AT%QMMQ1.10” “AT%MMMM1.7”
MS	MS<I/C(ok)n.m>	n=0-63 (node #), m=0-4095 (MI variable)	MACRO MstToSlv Read from or Write to MACRO MI variable INT48 (range 0-0xFFFFFFFF)	“AT%IMSI1.5” “AT%QMSI1.10” “AT%MMSI1.7” or “AT%IMS1.5” “AT%QMS1.10” “AT%MMS1.7”
MX	MX <I(ok)n.m>	n=0-63 (node #), m=0-65535 (Aux. Reg. Rd.) m=2-253 (Aux. Reg. Wrt.)	MACRO MstToSlv(Aux) Read from or Write to MACRO Aux. Reg. INT24 (range 0-65535)	“AT%IMXI1.5” “AT%QMXI1.10” “AT%MMXI1.7” or “AT%IMX1.5” “AT%QMX1.10” “AT%MMX1.7”
TWB	TWB<addr.offset.size.for mat>	See PMAC manual	Direct TWB address – REAL48	“AT%ITWB800” “AT%QTWB5” “AT%MTWB130”
TWD	TWD<addr.offset.size.dp. format>	See PMAC manual	Direct TWD address – REAL48	“AT%ITWD800” “AT%QTWD5” “AT%MTWD130”
TWR	TWR<addr.offset>	See PMAC manual	Direct TWR address – REAL48	“AT%ITWR8.0” “AT%QTWR4.4” “AT%MTWR254.7”
TWS	TWS<addr>	See PMAC manual	Direct TWS address – REAL48	“AT%ITWS8” “AT%QTWS5” “AT%MTWS13”

I/O Process Image Data Type Qualifier

The Process Image is where the Inputs are stored after input and the Outputs are stored before output. They exist as a data type in the IEC environment and the following table provides the possible data types and their bit width. PMAC LADDER supports all of these data types even though some may not be supported by functions.

Storage Format	IEC-1131 & PMAC Data Type	Process Image	Comment
Unsigned 8	BOOL, BYTE, USINT	X:mem 8 bits	
Signed 8	SINT	X:mem 8 bits	Sign extend before converting to PMAC data types
Unsigned 16	UINT, WORD	X:mem 16 bits	
Signed 16	INT	X:mem 16 bits	Sign extend before converting to PMAC data types
Signed 24	INT24	X:mem 24 bits	
Unsigned 32	DWORD, UDINT	L:mem 32 bits	
Signed 32	DINT	L:mem 32 bits	Sign extend before converting to PMAC data types
Long 48	REAL48	L:mem 48 bits	

Example PMAC Input & Output

VAR (* In IEC-1131 Free Variable Form Format *)

INPUTS ('I')

MyVarInp1 AT %I5 : INT; (* Convert I5 REAL48 to INT *)

MyVarInp2 AT %IM5 : REAL48;

MyVarInp3 AT %IP500 : DWORD; (* Convert P500 REAL48 to *)
(* DWORD *)

MyVarInp4 AT %IQ5 : DINT; (* Convert Q5 REAL48 to DINT *)

MyVarInp5 AT %IL5 : INT24; (* Convert L5 REAL48 to INT24 *)

MyVarInp6 AT %IX0x1234.4.16 : INT24;

(* Address=X:\$1234, offset=4, width =16, sign is determined by INT24 variable type. So the 16 bit data field will be sign extended. *)

MyVarInp7 AT %IX0x1234.4.16.0 : INT24;

(* Address=X:\$1234, offset=4, width =16, sign determined by INT24 is overridden by ".0". So the 16 bit data field will NOT be sign extended. *)

MyVarInp8 AT %IY0x14.4 : BOOL; (* Address=Y:\$14, Bit offset=4 *)

OUTPUTS ('O')

MyVarOut1 AT %QI5 : INT24; (* Convert INT24 to REAL48 I5 *)

MyVarOut2 AT %QM5 : REAL48;

MyVarOut3 AT %QP500 : DWORD;

(* Convert DWORD to REAL48 P500 *)

MyVarOut4 AT %QQ5 : DINT; (* Convert DINT to REAL48 Q5 *)

MyVarOut5 AT %QL5 : INT24; (* Convert INT24 to L5 REAL48 *)

MyVarOut6 AT %QX0x1234.4.16 : INT24;
(* The lower 16 bits are written to address=X:\$1234, offset=4, width =16 *)

MyVarOut7 AT %QY0x14.4 : BOOL;
(* Bit 0 is Address=Y:\$14, Bit offset=4 *)

INPUTS & OUTPUTS ('M')

The Input & Output would have the same process memory location.

MyVarInpOut1 AT %MI5 : INT;
MyVarInpOut2 AT %MM5 : REAL48;
MyVarInpOut3 AT %MP500 : DWORD;
MyVarInpOut4 AT %MQ5 : DINT;
MyVarInpOut5 AT %ML5 : INT24;
MyVarInpOut6 AT %MX0x1234.4.16.1.0 : INT24;
MyVarInpOut7 AT %MY0x14.4 : BOOL;

END_VAR